

Paragon Insights Data Ingest Guide

Published
2023-10-03

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

Juniper Networks, the Juniper Networks logo, Juniper, and Junos are registered trademarks of Juniper Networks, Inc. in the United States and other countries. All other trademarks, service marks, registered marks, or registered service marks are the property of their respective owners.

Juniper Networks assumes no responsibility for any inaccuracies in this document. Juniper Networks reserves the right to change, modify, transfer, or otherwise revise this publication without notice.

Paragon Insights Data Ingest Guide

Copyright © 2023 Juniper Networks, Inc. All rights reserved.

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

About This Guide | vi

1

“Push” Model Data Ingest Methods

Paragon Insights Push-Model Overview | 2

Paragon Insights Push-Model Ingest Methods | 2

Native GPB | 3

NetFlow | 4

Example: Add a Device In Paragon Insights, Configure Paragon Insights for NetFlow, and Monitor | 8

Add the Device In Paragon Insights | 8

Add Device Group | 9

Define NetFlow Ingest Settings - Review Predefined Templates | 9

Define NetFlow Ingest Settings - (Optional) Create Your Own NetFlow Template | 10

Define NetFlow Ingest Settings - Clone an Existing NetFlow Template | 11

Define NetFlow Ingest Settings - Delete a NetFlow Template | 11

Configure a Rule Using the Flow Sensor | 12

Add the Rule to a Playbook | 20

Apply the Playbook to a Device Group | 20

Monitor the Devices | 21

Differences Between NetFlow and sFlow | 21

sFlow | 22

OpenConfig | 35

Syslog | 38

Monitor the Devices | 59

SNMP Trap and Inform Notifications | 59

Glossary | 60

Configurations | 62

Understand Inband Flow Analyzer 2.0 | 73

- Configure Device Details for Inband Flow Analyzer Devices | 79**
- Delete an Inband Flow Analyzer Device | 80**
- Understand Bring Your Own Ingest | 81**
- Bring Your Own Ingest Default Plug-in Workflow | 83**
- Load Bring Your Own Ingest Default Plug-ins | 85**
- Configure Bring Your Own Ingest Default Plug-in Instances | 86**
- Configure Ingest Mapping for Default BYOI Plug-ins | 88**
- Bring Your Own Ingest Custom Plug-in Workflow | 89**
- Build and Load BYOI Custom Plug-in Images | 91**
 - Use JSON Configuration File Attributes in Ingest Image | 93
 - Create a Shell Script for Configuration Updates | 96
 - Tag and Export the BYOI Custom Plugin Image | 96
 - Configure Kubernetes YAML File | 96
 - (Optional) Assign Virtual IP Address to Plugin | 100
 - Load the BYOI Custom Plugin | 102
- Configure Bring Your Own Ingest Custom Plug-in Instances | 102**
- Use the Sample Rule and Playbook Configurations for BYOI Custom Plug-ins | 105**
- Delete a Bring Your Own Ingest Plug-in | 106**

2

“Pull” Model Data Ingest Methods

- Paragon Insights Pull-Model Overview | 109**
- Paragon Insights Pull-Model Ingest Methods | 109**
 - Server Monitoring Ingest | 109
 - Configure a Rule Using Server Monitoring Sensor | 114
 - Understanding kube-state-metrics Service | 116
 - iAgent (CLI/NETCONF) | 128
 - Example: PaloAlto Panos- Show Running Security Policy | 131

Outbound SSH (Device-Initiated) | 135

iAgent - vCenter/ESXi Server Monitor | 142

SNMP | 142

SNMP in Paragon Insights | 143

Example: Creating a Rule using SNMP Ingest | 157

CONFIGURE NETWORK DEVICES | 158

CREATE RULE, APPLY PLAYBOOK | 158

Monitor the Devices | 171

About This Guide

Paragon Insights (formerly HealthBot) accepts data from a variety of Juniper and third-party devices. It accepts this data from various types of telemetry sensors and from traditional network management protocols like syslog and SNMP. We provide this data ingest guide as a way to understand the ingest methods that Paragon Insights supports and so that you can decide the best ways for you to get the needed health and performance data from your devices.

As mentioned in the user guide, Paragon Insights supports push and pull models of data collection. In the push model, your devices push telemetry data to Paragon Insights. In the pull mode, Paragon Insights periodically polls your devices for data. This guide describes each of the supported ingest methods, with examples, sorted by whether they fall into the push or pull model. Along with each description, we provide the required Junos OS version and device configurations needed to enable the specific ingest type.

1

CHAPTER

“Push” Model Data Ingest Methods

- [Paragon Insights Push-Model Overview | 2](#)
 - [Paragon Insights Push-Model Ingest Methods | 2](#)
 - [Understand Inband Flow Analyzer 2.0 | 73](#)
 - [Configure Device Details for Inband Flow Analyzer Devices | 79](#)
 - [Delete an Inband Flow Analyzer Device | 80](#)
 - [Understand Bring Your Own Ingest | 81](#)
 - [Bring Your Own Ingest Default Plug-in Workflow | 83](#)
 - [Load Bring Your Own Ingest Default Plug-ins | 85](#)
 - [Configure Bring Your Own Ingest Default Plug-in Instances | 86](#)
 - [Configure Ingest Mapping for Default BYOI Plug-ins | 88](#)
 - [Bring Your Own Ingest Custom Plug-in Workflow | 89](#)
 - [Build and Load BYOI Custom Plug-in Images | 91](#)
 - [Configure Bring Your Own Ingest Custom Plug-in Instances | 102](#)
 - [Use the Sample Rule and Playbook Configurations for BYOI Custom Plug-ins | 105](#)
 - [Delete a Bring Your Own Ingest Plug-in | 106](#)
-

Paragon Insights Push-Model Overview

As the number of objects in the network, and the metrics they generate, have grown, gathering operational statistics for monitoring the health of a network has become an ever-increasing challenge. Traditional 'pull' data-gathering models, like SNMP and the CLI, require additional processing to periodically poll the network element, and can directly limit scaling.

The 'push' model overcomes these limits by delivering data asynchronously, which eliminates polling. With this model, the Paragon Insights server can make a single request to a network device to stream periodic updates. As a result, the 'push' model is highly scalable and can support the monitoring of thousands of objects in a network. Junos devices support this model in the form of the [Junos Telemetry Interface \(JTI\)](#).

Paragon Insights currently supports five 'push' ingest types.

Paragon Insights Push-Model Ingest Methods

IN THIS SECTION

- [Native GPB | 3](#)
- [NetFlow | 4](#)
- [Example: Add a Device In Paragon Insights, Configure Paragon Insights for NetFlow, and Monitor | 8](#)
- [Differences Between NetFlow and sFlow | 21](#)
- [sFlow | 22](#)
- [OpenConfig | 35](#)
- [Syslog | 38](#)
- [Monitor the Devices | 59](#)
- [SNMP Trap and Inform Notifications | 59](#)

Paragon Insights currently supports the following push-model sensors:

- ["Native GPB" on page 3](#)
- ["NetFlow" on page 4](#)

- ["sFlow" on page 22](#)
- ["OpenConfig" on page 35](#)
- ["Syslog" on page 38](#)
- ["SNMP Trap and Inform Notifications" on page 59](#)

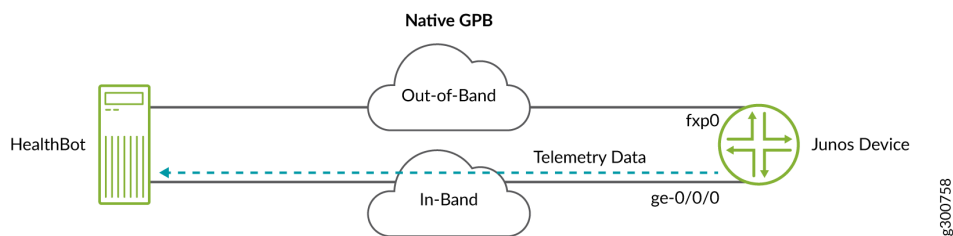
Native GPB

IN THIS SECTION

- [Native GPB - Device Configuration | 3](#)

Native sensors use a Juniper-proprietary data model using Google protocol buffers (GPB). The device pushes telemetry data (when configured) over UDP.

The device pushes data from the Packet Forwarding Engine, that is, directly from a line card. This means telemetry data is sent over the forwarding plane, so the collector must have in-band connectivity to the device.



To use native format, you configure the device with settings that include where to send the telemetry data. When you configure Paragon Insights to start collecting the data, the stream is already flowing towards the server.

For more information on native sensors, see [Understanding the Junos Telemetry Interface Export Format of Collected Data](#).

Native GPB - Device Configuration

The requirements for using the Native GPB sensor type on a Junos OS device are shown below.

- Junos OS Version: 15.1 or later
- Required configuration—configure a sensor profile for each relevant related rule in Paragon Insights:

```
##Streaming Server Profile
set services analytics streaming-server COLLECTOR-1 remote-address <HealthBot-server-address>
set services analytics streaming-server COLLECTOR-1 remote-port 22000
##Export Profile
set services analytics export-profile EXP-PROF-1 local-address <local-router-IP>
set services analytics export-profile EXP-PROF-1 local-port 22001
set services analytics export-profile EXP-PROF-1 reporting-rate 30
set services analytics export-profile EXP-PROF-1 format gpb
set services analytics export-profile EXP-PROF-1 transport udp
##Sensor Profile
set services analytics sensor SENSOR-1 server-name COLLECTOR-1
set services analytics sensor SENSOR-1 export-name EXP-PROF-1
set services analytics sensor SENSOR-1 resource <resource> # example /junos/system/linecard/
interface/
```

To configure streaming server port in Paragon Insights GUI:

1. Go to **Settings > Ingest**.
2. Select **Native GPB** tab on the Ingest Settings page.
3. Enter the port number. The port number must be the same as the remote-port configured in the streaming server profile.

You can use the toggle button to enable or disable the **Port** field.

4. Click **Save & Deploy** to enable the sensor to collect data in your network.

See [Configuring a Junos Telemetry Interface Sensor](#) for more information.

NetFlow

IN THIS SECTION

- [NetFlow Templates | 5](#)
- [NetFlow Ingest Processing | 6](#)

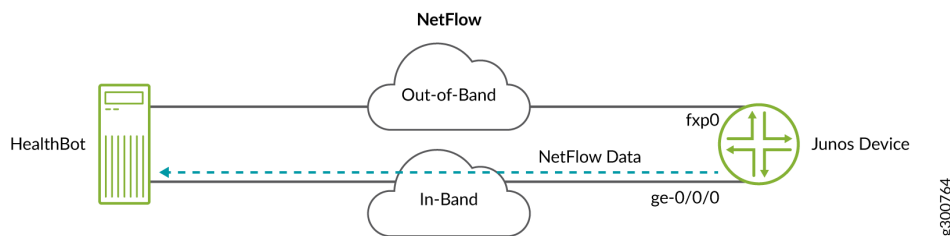
● NetFlow - Device Configuration | 7

Starting with Release 3.0.0, Paragon Insights (formerly HealthBot) supports NetFlow natively as another ingest method, using a data model that aligns with other Paragon Insights ingest mechanisms to provide all the same feature richness. With this release, Paragon Insights supports NetFlow v9 and NetFlow v10 (IPFIX).

How it Works

NetFlow is a network protocol for collecting IP traffic statistics, which can then be exported to a tool for analysis. NetFlow is available in different versions, the latest being NetFlow v9 and NetFlow v10. The NetFlow v9 data export format is described in [RFC 3954](#); NetFlow v10 is officially known as IPFIX and standardized in [RFC 7011](#).

Junos devices support flow monitoring and aggregation using these protocols; the Junos OS samples the traffic, builds a flow table, and sends the details of the flow table over a configured UDP port to a collector, in this case Paragon Insights. Paragon Insights receives the incoming Netflow data, auto-detects it as v9 or v10, and process it further.



As shown above, the network device pushes data from the Packet Forwarding Engine, that is, directly from a line card. This means flow data is sent over the forwarding plane, so the collector must have in-band connectivity to the device. To use the flow sensor option, you configure the device with settings that include where to send the flow data. When you configure Paragon Insights to start collecting the data, the flow data is already flowing towards the server.

NetFlow Templates

Where other ingest methods have established sensor formats and identification details - for example, native GPB references paths, SNMP references MIBs, etc. - flow has no equivalent mechanism. Instead, Paragon Insights uses templates. These flow templates provide a mechanism to identify and decode incoming flow data before sending it for further processing.

Paragon Insights provides predefined flow templates for NetFlow v9 and v10 (IPFIX), or you can define your own. The predefined templates match those which the Junos OS currently supports. For example,

the Junos OS template, `ipv4-template`, aligns with the Paragon Insights template `hb-ipfix-ipv4-template`. To view the fields used in the Junos OS templates, see [Understanding Inline Active Flow Monitoring](#).

NOTE: In the current ingest implementation for NetFlow, the following field types are not supported:

- Fields for enterprise specific elements
- Variable length fields

NetFlow Ingest Processing

The raw flow data that Paragon Insights receives is in binary format and unreadable. In order to make this data usable, Paragon Insights processes the incoming flow data as follows:

- Paragon Insights listens for incoming flow data on a configured port
- Since NetFlow messages don't include a field that identifies the sending device, Paragon Insights uses the configured source IP address to derive a device ID
- Templates identify and decode incoming flow data to determine which fields it contains

The resulting decoded and normalized data is now in a readable and usable format. Here is an example of flow data decoded using the `hb-ipfix-ipv4-template` template:

```
hb-ipfix-ipv4-template, destinationIPv4Address=192.168.48.200, destinationTransportPort=443,
icmpTypeCodeIPv4=0, ingressInterface=1113, ipClassOfService=0, protocolIdentifier=6,
sourceIPv4Address=172.16.235.191, sourceTransportPort=51032,
bgpDestinationAsNumber=4200000000i, bgpSourceAsNumber=65000i, destinationIPv4PrefixLength=24i,
dot1qCustomerVlanId=0i, dot1qVlanId=0i, egressInterface=1041i, flowEndMilliseconds=1483484591502u,
flowEndReason=2i, flowStartMilliseconds=1483484577531u, ipNextHopIPv4Address="192.168.41.49",
maximumTTL=120i, minimumTTL=120i, octetDeltaCount=188i, packetDeltaCount=4i, sourceIPv4PrefixLength=19i,
tcpControlBits=16i, vlanId=0i 1483484642000000000
```

The data shows information from the NetFlow messages using naming according to [IPFIX Information Elements](#). For example, `destinationIPv4Address` maps to element ID 12 in the elements table.

- Paragon Insights then performs further tagging, normalization, and aggregation as defined in the corresponding rule by the user.
- Finally, the time-series database, TSDB receives the data. This is where things like trigger evaluation happen.



WARNING: For NetFlow ingest, ensure that there is no source NAT in the network path(s) between the device and Paragon Insights. If the network path contains source NAT, then the received device information is not accurate.

NetFlow - Device Configuration

To use NetFlow as an ingest method in Paragon Insights, you must add configuration to the device you wish to monitor, to enable it to export flow data into Paragon Insights.

This example includes the Netflow v10 IPv4 template; adjust as needed for your environment. If not already done, complete device configuration to send NetFlow data to Paragon Insights as shown below.

IPFIX template configuration

```
set services flow-monitoring version-ipfix template IPv4-TEMPLATE ipv4-template
```

Apply IPFIX template to enable traffic sampling

```
set forwarding-options sampling instance IPFIX-IPv4-INSTANCE input rate 10
set forwarding-options sampling instance IPFIX-IPv4-INSTANCE family inet output flow-server
10.102.70.200 port 2055
```

10.102.70.200 = Paragon Insights server

NOTE: This is the IP address of Paragon Insights node receiving the NetFlow traffic as per the Device Group configuration field of **Flow Deploy Nodes**. This is not the virtual IP address of the Paragon Insights cluster (Netflow protocol has limitation in receiving the traffic using virtual IP address).

port 2055; use this value in Paragon Insights GUI (device group config)

```
set forwarding-options sampling instance IPFIX-IPv4-INSTANCE family inet output flow-server
10.102.70.200 version-ipfix template IPv4-TEMPLATE
set forwarding-options sampling instance IPFIX-IPv4-INSTANCE family inet output inline-jflow
source-address 198.51.100.1
```

inline-jflow = Enable inline flow monitoring for traffic from the designated address

198.51.100.1 = in-band interface doing the exporting; use this value in Paragon Insights GUI (device config)

Associate sampling instance with the FPC

```
set chassis fpc 0 sampling-instance IPFIX-IPv4-INSTANCE
```

Specify which interface traffic to sample

```
set interfaces ge-0/0/0 unit 0 family inet sampling input
set interfaces ge-0/0/0 unit 0 family inet sampling output
```

Example: Add a Device In Paragon Insights, Configure Paragon Insights for NetFlow, and Monitor

IN THIS SECTION

- [Add the Device In Paragon Insights | 8](#)
- [Add Device Group | 9](#)
- [Define NetFlow Ingest Settings - Review Predefined Templates | 9](#)
- [Define NetFlow Ingest Settings - \(Optional\) Create Your Own NetFlow Template | 10](#)
- [Define NetFlow Ingest Settings - Clone an Existing NetFlow Template | 11](#)
- [Define NetFlow Ingest Settings - Delete a NetFlow Template | 11](#)
- [Configure a Rule Using the Flow Sensor | 12](#)
- [Add the Rule to a Playbook | 20](#)
- [Apply the Playbook to a Device Group | 20](#)
- [Monitor the Devices | 21](#)

The following example walks through how to:

Add the Device In Paragon Insights

Now add the device in Paragon Insights, specifying the IP address(es) that will send the flow data.

1. In the Paragon Insights GUI, click **Configuration > Device** in the left-nav bar, and click the add device button (+ Device).

2. Click the + **(Add Device)** button
3. In the **Add Device(s)** window that appears, fill in the appropriate fields.
Be sure to fill in the **Flow IPs** field with the IP address(es) from which NetFlow data will arrive.
4. Click **Save & Deploy**.

For more information about adding a device, see *Adding a Device* in *Manage Devices, Device Groups, and Network Groups*.

Usage Notes:

- Incoming NetFlow messages don't include a device ID; Paragon Insights uses the message's source IP address to derive a device ID
- When configuring this step, use the in-band interface IP address you configured in the sampling instance configuration on the device.

Add Device Group

With the device added, you now need to create a device group and define the flow ingest port for the device group.

1. Click **Configuration > Device Group**
2. Click the + **(Add Device Group)** button
3. In the Add Device Group window that appears, fill in the fields as appropriate. In the **Flow Ports** field, enter the port(s) on which NetFlow data will arrive.

NOTE: If your Paragon Insights installation is a multi-node installation using Kubernetes, you must also specify which Paragon Insights nodes will receive the NetFlow traffic by filling in the **Flow Deploy Nodes** field in the **Add/Edit Device Group** window.

4. Click **Save & Deploy**

Usage Notes:

- Paragon Insights will listen for NetFlow messages on this port for devices in this group.
- The configured NetFlow ingest ports cannot be the same across device groups. You must configure a different port (or ports) for each group.

Define NetFlow Ingest Settings - Review Predefined Templates

NetFlow templates provide a mechanism to identify and decode incoming flow data before sending it for further processing within Paragon Insights.

1. Click **Settings > Ingest** in the left-nav bar.

The **Ingest Settings** page appears.

2. Click the **NetFlow** tab to view the **NetFlow Settings** page.
3. On the NetFlow settings page, review the available templates for use in a rule.

Usage Notes:

- Notice that there are default flow templates for IPv4, IPv6, MPLS, MPLS-IPv4, MPLS-IPv6, and VPLS, for each of NetFlow v9 and v10.
- The NetFlow templates include recognition patterns, called include fields and exclude fields, which help to recognize, identify, and categorize the incoming messages.
- Since NetFlow messages don't distinguish between keys and values (all fields are simply incoming data), the templates specify which fields should be treated as keys for raw data.

Define NetFlow Ingest Settings - (Optional) Create Your Own NetFlow Template

If the existing templates do not meet your needs, you can create your own template. You can also use custom templates to support other vendors' devices.

1. Click the plus (+) icon on the NetFlow settings page. .
2. In the Add Template window that appears, fill in the following fields (you can leave the other settings as is):
 - Template Name - give the template a name
 - NetFlow version - select v9 or v10
 - Priority - Available values are 1 through 10
 - Include Fields - add one or more fields that you want included in the template you wish to use
 - Exclude Fields - add one or more fields that you do not want included in the template you wish to use
 - Key Fields - specify which fields in the incoming messages should be treated as keys
3. Click **Save & Deploy**

You should now see the template added to the NetFlow settings page.
4. (Optional) Repeat the steps above to create more templates.

Usage Notes:

- Priority - when a playbook includes multiple rules using the flow sensor, the priority value identifies which sensor and template gets priority over the other(s).
- Include/Exclude fields - include fields to help identify the template to use, or at least a 'short list' of templates to use; exclude fields then narrow down to the single desired template.

- Example 1 - consider the *hb-ipfix-ipv4-template* template: it includes two IPv4 fields to narrow down to *hb-ipfix-ipv4-template* and *hb-ipfix-mpls-ipv4-template*, and excludes an MPLS field to eliminate *hb-ipfix-mpls-ipv4-template*, leaving only *hb-ipfix-ipv4-template*.
- Example 2 - consider the *hb-ipfix-mpls-ipv4-template* template: it includes the same two IPv4 fields to narrow down to *hb-ipfix-ipv4-template* and *hb-ipfix-mpls-ipv4-template*. It also includes an MPLS field, which immediately eliminates the former template and leaving the latter as the template to use.

Define NetFlow Ingest Settings - Clone an Existing NetFlow Template

Starting in Paragon Insights Release 4.0.0, you can clone an existing NetFlow template.

To clone an existing NetFlow template:

1. Click **Settings > Ingest** from the left-nav bar.
The **Ingest Settings** page is displayed.
2. Click the **NetFlow** tab to view the **Netflow Settings** page.
3. To clone a particular template for Paragon Insights Releases 4.1.0 and 4.0.0, click the **Clone** icon.

To clone a particular template for Paragon Insights Release 4.2.0 and later, select the option button next to the name of the template and click **Clone**.

The Clone Template: *<name of template>* page is displayed.

From the **Clone Template: <name of template>** page, you can

- Edit the **Template Name**, **Description**, and **Priority** sections.
 - Choose between **Netflow v9** or **Netflow v10** versions.
 - Add or exclude fields from **Include Fields**, **Exclude Fields**, and **Key Fields**.
4. After you have made the necessary edits, click **Save** to save the modifications and to clone the template.

Alternatively, click **Save & Deploy** to save modifications, clone the template, and deploy the template.

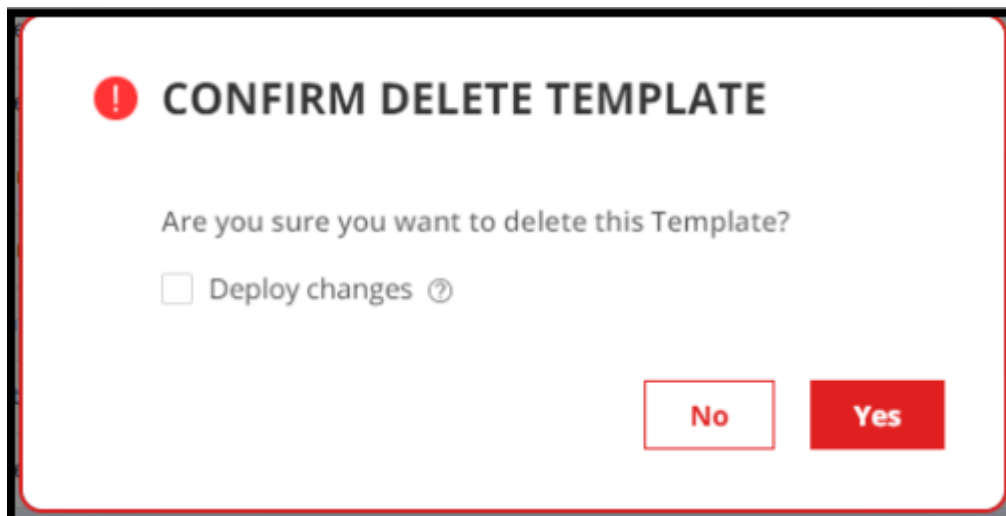
Define NetFlow Ingest Settings - Delete a NetFlow Template

To delete a NetFlow template:

1. Click **Settings > Ingest** from the left-nav bar.
The **Ingest Settings** page is displayed.
2. Click the **NetFlow** tab to view the **NetFlow** page.
3. Select the device that you want to delete, and click the **delete (trash can)** icon.

The **CONFIRM DELETE TEMPLATE** pop-up appears.

Figure 1: Confirm Delete Template Pop-up



4. Do any one of the following:

- Click **Yes** to delete the NetFlow template from the database. However, the changes are not applied to the ingest service.

NOTE:

- We recommended that you do not delete a NetFlow template that is currently in use.
 - After you delete a NetFlow template from the database, you cannot associate that template with another device group or device even if you have not deployed changes.
 - You can also deploy changes to the ingest service or roll back the changes that you have already deleted, from the **PENDING CONFIGURATION** page. For more information, see *Commit or Roll Back Configuration Changes in Paragon Insights*.
- Select the **Deploy changes** check box and then click **Yes** to delete the template from the database, and to apply the changes to the ingest service.
 - (Optional) Click **No** to cancel this operation.

Configure a Rule Using the Flow Sensor

With the NetFlow ingest settings complete, you can now create a rule using flow as the sensor.

This example rule includes three elements:

- A flow sensor that uses the NetFlow v10 IPv4 template
- Six fields capturing data of interest
- A trigger that indicates when traffic flow is higher or lower than expected

NOTE: See the usage notes at the end of this section for more detail on what has been configured.

1. Click **Configuration > Rules** in the left-nav bar.
2. On the Rules page, click the **+ Add Rule** button.
The Rules page refreshes to show a nearly empty rule on the right part of the page.
3. In the top row of the rule window, leave the topic set as *external* and set the rule name that appears after the slash (/). In this example, it is *periodic-aggregation-flow-rule*.
4. Add a description and synopsis if you wish.
5. Click the **+ Add Sensor** button and enter the following parameters in the Sensors tab:

The screenshot shows the 'Sensors' configuration page. At the top, there are tabs for 'Sensors', 'Fields', 'Vectors', 'Variables', 'Functions', 'Triggers', and 'Rule Properties'. The 'Sensors' tab is active. A blue '+ ADD SENSOR' button is in the top left. Below it, a card for 'ipv4-flow-sensor' is shown. To the right of the card is a 'DELETE IPV4-FLOW-SENSOR' button. The configuration fields are:

- Sensor Name ***: A text input field containing 'ipv4-flow-sensor'.
- Sensor Type**: A dropdown menu with 'Flow' selected.
- Template Name ***: A dropdown menu with 'hb-ipfix-ipv4-template' selected.

6. Now move to the Fields tab, click the **+ Add Field** button and enter the following parameters to configure the first field, *source-ipv4-address*:

Sensors Fields Vectors Variables Functions Triggers Rule Properties

+ ADD FIELD

source-ipv4-address DELETE SOURCE-IPV4-ADDRESS

Field Name * ?
source-ipv4-address

Description ?
Source IPv4 address

Field Type
string

Add to Rule Key ?

Ingest type (Field source)
Sensor

Sensor Path * Data if missing
ipv4-flow-sensor sourceIPv4Address Zero suppression Default value

Where (filter using expression)
+ ADD EXPRESSION

7. Click the **+ Add Field** button again and enter the following parameters to configure the second field, *destination-ipv4-address*:

Sensors Fields Vectors Variables Functions Triggers Rule Properties

+ ADD FIELD

source-ipv4-address DELETE DESTINATION-IPV4-ADDRESS

destination-ipv4-address

Field Name * ?
destination-ipv4-address

Description ?
Destination IPv4 Address

Field Type
string

Add to Rule Key ?

Ingest type (Field source)
Sensor

Sensor Path * Data if missing
ipv4-flow-sensor destinationIPv4Address Zero suppression Default value

Where (filter using expression)
+ ADD EXPRESSION

8. Click the **+ Add Field** button again and enter the following parameters to configure the third field, *sensor-traffic-count*:

Sensors **Fields** Vectors Variables Functions Triggers Rule Properties

+ ADD FIELD **DELETE SENSOR-TRAFFIC-COUNT**

source-ipv4-address
destination-ipv4-address
sensor-traffic-count
total-traffic-count

Field Name * ?
sensor-traffic-count

Description ?
Sensor octet count for IPv4 traffic measurement

Field Type
integer

Add to Rule Key ?

Ingest type (Field source)
Sensor

Sensor **Path *** **Data if missing**
ipv4-flow-sensor octetDeltaCount Zero suppression Default value

Where (filter using expression)
+ ADD EXPRESSION

9. Click the **+ Add Field** button again and enter the following parameters to configure the fourth field, *total-traffic-count*:

Sensors **Fields** Vectors Variables Functions Triggers Rule Properties

+ ADD FIELD **DELETE TOTAL-TRAFFIC-COUNT**

source-ipv4-address
destination-ipv4-address
sensor-traffic-count
total-traffic-count

Field Name * ?
total-traffic-count

Description ?
Periodic sum of packet count for IPv4 measured in device

Field Type
integer

Add to Rule Key ?

Ingest type (Field source)
Formula

Formula **Field ***
Sum \$sensor-traffic-count

Time range *
10s

10. Click the **+ Add Field** button again and enter the following parameters to configure the fifth field, *traffic-count-maximum*:

The screenshot shows the 'Fields' configuration page with the following settings for the 'traffic-count-maximum' field:

- Field Name:** traffic-count-maximum
- Description:** Maximum total traffic count
- Field Type:** integer
- Add to Rule Key:**
- Ingest type (Field source):** Constant
- Constant value:** {{traffic-count-max}}

11. Click the **+ Add Field** button once more and enter the following parameters to configure the sixth field, *traffic-count-minimum*:

The screenshot shows the 'Fields' configuration page with the following settings for the 'traffic-count-minimum' field:

- Field Name:** traffic-count-minimum
- Description:** Minimum total traffic count
- Field Type:** integer
- Add to Rule Key:**
- Ingest type (Field source):** Constant
- Constant value:** {{traffic-count-min}}

12. As the last step for the fields configuration, set the field aggregation time-range value to 10s:

The screenshot shows the 'Field aggregation time-range' setting set to 10s.

13. Now move to the Variables tab, click the **+ ADD VARIABLE** button and create the *traffic-count-max* and *traffic-count-min* variables that are the constants for the *traffic-count-maximum* and *traffic-count-minimum* fields, respectively.

Sensors Fields Vectors **Variables** Functions Triggers Rule Properties

+ ADD VARIABLE

traffic-count-max
traffic-count-min

DELETE TRAFFIC-COUNT-MAX

Variable name * ?
traffic-count-max

Default Value ?
10000

Type * ?
Integer

Description ?
Maximum traffic count threshold in PPS

NOTE: Only the definition for the *traffic-count-max* is shown graphically. Choose an appropriate **Default Value** when configuring both *traffic-count-max* and *traffic-count-min* variables. The value shown above is for testing purposes only and may not be appropriate for your network.

14. Now move to the Triggers tab, click the **+ Add trigger** button and enter the following parameters to configure a trigger called *traffic-measurement-trigger*.

Sensors Fields Vectors Variables Functions **Triggers** Rule Properties

+ ADD TRIGGER

traffic-measurement-trigger

DELETE TRAFFIC-MEASUREMENT-TRIGGER

Trigger Name * ?
traffic-measurement-trigger

Frequency ?
90s

Disable alert deduplication

Term traffic-anomaly-gr **X**

WHEN

Left operand	Operator	Right operand	All in time range
\$total-traffic-count	>	Traffic-count-maximum	Enter a time range -

+ ADD CONDITION

THEN

Color
■

Message
Total traffic count is above normal. Current total traffic count is \$total-traffic-count.

traffic-measurement-trigger

Trigger Name * [?]

traffic-measurement-trigger

Frequency [?]

90s

Disable alert deduplication

Term traffic-abnormal-gr ✕

Term traffic-abnormal-ls ✕

WHEN

Left operand	Operator	Right operand	All in time range
\$total-traffic-count	<	\$traffic-count-minimum	Enter a time range ⊖

[+ ADD CONDITION](#)

THEN

Color

Message

Total traffic count is below normal. Current total traffic count is \$total-traffic-count.

Evaluate next term

traffic-measurement-trigger

Trigger Name * [?]

traffic-measurement-trigger

Frequency [?]

90s

Disable alert deduplication

Term traffic-abnormal-gr ✕

Term traffic-abnormal-ls ✕

Term default-term ✕

WHEN

[+ ADD CONDITION](#)

THEN

Color

Message

Total traffic count is normal. Current total traffic count is \$total-traffic-count.

Evaluate next term

15. At the upper right of the window, click the **Save & Deploy** button.

Usage Notes:

- **Sensor Tab:**
 - The sensor name *ipv4-flow-sensor* is user-defined

- The sensor type is flow
- The sensor uses the predefined template *hb-ipfix-ipv4-template*
- **Variables Tab:**
 - The variables *traffic-count-max* and *traffic-count-min* are statically configured integers. In this case the values represent Bytes per second
 - These values are referenced in fields *traffic-count-maximum* and *traffic-count-minimum* and provide a reference point to compare against the *total-traffic-count* field
- **Fields Tab:**
 - Six fields are defined; some fields are used in the trigger settings while one field is referenced within another field
 - The field names are user-defined fields (UDF)
 - Fields *source-ipv4-address*, *destination-ipv4-address*, and *sensor-traffic-count* are extracting information from the flow sensor input
 - Path values for these fields identify specific values from the NetFlow messages, using naming according to [IPFIX Information Elements](#)
 - Fields *source-ipv4-address* and *destination-ipv4-address* have the Add to rule key setting enabled, indicating that this field should be shown as a searchable key for this rule on the device health pages
 - Field *total-traffic-count* - sums the IPv4 packet count from the *sensor-traffic-count* field every 10 seconds
 - The fields *traffic-count-maximum* and *traffic-count-minimum* are simply fixed values; the values are derived from the variables defined above
 - Field *aggregation time-range* - typically set to a value higher (longer) than individual field time range settings with the aim of reducing the frequency of information being sent to the database
- **Triggers Tab:**
 - The trigger name *traffic-measurement-trigger* is user-defined.
 - *frequency 90s* - HearthBot compares traffic counts every 90 seconds
 - In the term *traffic-abnormal-gr*:
 - When *\$total-traffic-count* (the periodic count of incoming IPv4 traffic) is greater than *\$traffic-count-maximum* (2500 Bps), show red and the message: "Total traffic count is above normal. Current total traffic count is : *\$total-traffic-count*".

- In the term `traffic-abnormal-ls`:
 - When `$total-traffic-count` (the periodic count of incoming IPv4 traffic) is less than `$traffic-count-minimum` (500 Bps), show yellow and the message: "Total traffic count is below normal. Current total traffic count is : `$total-traffic-count`".
- In the term `default-term`:
- Otherwise, show green and the message: "Total traffic count is normal. Current total traffic count is : `$total-traffic-count`".

Add the Rule to a Playbook

With the rule created, you can now add it to a playbook. For this example, create a new playbook to hold the new rule.

1. Click **Configuration > Playbooks** in the left-nav bar.
2. On the Playbooks page, click the create **+ Create Playbook** button.
3. On the page that appears, enter the following parameters:

The screenshot shows a 'Create Playbook' form with the following fields and values:

- Name ***: periodic-aggregation-flow-playbook
- Synopsis**: Flow playbook with periodic aggregation
- Rules ***: external/periodic-aggregation-flow-rule

Buttons at the bottom: CANCEL, SAVE, SAVE & DEPLOY.

4. Click **Save & Deploy**

Apply the Playbook to a Device Group

1. On the Playbooks page, click the **Apply (Airplane)** icon for the playbook you configured above.
2. On the Run Playbook page that appears
 - Enter a name for the playbook instance.
 - Select the desired device group from the **Apply Group** pull-down menu.

- Click Run Instance.
3. On the Playbooks page, confirm that the playbook instance is running. Note that the playbook may take some time to activate.

Monitor the Devices

With the playbook applied, you can begin to monitor the devices.

1. Click **Monitor > Device Group Health** in the left-nav bar and select the device group to which you applied the playbook from the **Device Group** pull-down menu.
2. Select one or more of the devices to monitor.
3. In the Tile View, the external tile contains the parameters from the rule you configured earlier.

Differences Between NetFlow and sFlow

While the names sound similar, NetFlow and sFlow are two very different protocols. [Table 1 on page 21](#) shows the differences between "[NetFlow](#)" on page 4 and "[sFlow](#)" on page 22.

Table 1: NetFlow and sFlow Differences

	NetFlow/IPFIX	sFlow
Overview	A flow is a sequence of packets that share the same properties and travels between a sending host and a receiving host. Flow analysis looks at the metadata in the flow of packets. NetFlow is based on flow analysis techniques.	sFlow takes actual network packet samples from a device and forwards them to a collector for analysis. sFlow is a packet sampling and analysis technology, not a flow analysis technology.
More Detail	Flow analysis deals with identifying top talkers, top protocols, bandwidth usage, etc. within the network. It uses traffic metadata to provide this information.	Packet analysis deals with gaining in-depth information regarding a specific network conversation. This is possible because the packet header and payload are included in the sFlow data.

Table 1: NetFlow and sFlow Differences (Continued)

	NetFlow/IPFIX	sFlow
Data	<p>NetFlow/IPFIX allows a user to define what data is collected from the devices as fields in a template. The data can be based on packet headers, traffic characteristics, or values from within the devices themselves.</p> <p>NetFlow can provide information from layer 2 through layer 4 of the OSI model.</p>	<p>sFlow specifies a single way to report data: provide the entire packet header and payload data from a sampled interface.</p> <p>sFlow can provide information from layer 2 through layer 7 of the OSI model.</p>
Time	<p>NetFlow/IPFIX headers contain the export time of the flow. Flow data can accumulate and be sent at any configured frequency. Template definitions allow for over 30 different methods to represent the time of an observed flow.</p>	<p>The sFlow standard requires that sampled packets are forwarded to the collector immediately upon capture.</p>
State	<p>NetFlow/IPFIX is a stateful protocol. The device receiving NetFlow data (Paragon Insights) must confirm the receipt of the data before more is sent.</p>	<p>sFlow is a stateless protocol. The sender simply forwards captured packets to the receiver (Paragon Insights).</p>
Processing	<p>NetFlow/IPFIX is a stateful protocol. The sending and receiving devices must both have the correct NetFlow/IPFIX template to encode and decode the flow. New templates can be configured by users and shared between sender and receiver (Paragon Insights)</p>	<p>The sending device uses the sFlow standard to encode the messages (packets) and the collector (Paragon Insights) uses the same standard to decode them.</p>

sFlow

IN THIS SECTION

- [sFlow Protocol | 23](#)
- [sFlow Ingest Processing | 23](#)
- [sFlow - Configuration in Paragon Insights | 24](#)

Starting with Paragon Insights Release 3.2.0, Paragon Insights supports sFlow (v5) natively as another flow-based ingest method, using a data model that aligns with other Paragon Insights ingest mechanisms to provide all the same feature richness.

sFlow Protocol

sFlow is a statistical sampling-based technology for high-speed switched or routed networks. You can configure sFlow to continuously monitor traffic at wire speed on all interfaces simultaneously if you want.

sFlow provides or helps with:

- detailed and quantitative traffic measurements at gigabit speeds
- insight into forwarding decisions
- troubleshooting for network issues
- congestion control
- security and audit trail analysis
- route profiling

Everything that sFlow does above, it does without impact to forwarding or network performance. For more information on sFlow, see: [RFC 3176](#), [InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks](#).

As a statistical sampling protocol, Juniper's sFlow agent samples the traffic and counters on network interfaces, creates sFlow datagrams and forwards them to external sFlow collectors. Paragon Insights is one such collector.

sFlow Ingest Processing

The raw sFlow data that Paragon Insights receives is in binary format and unreadable. In order to make this data usable, Paragon Insights processes the incoming flow data as follows:

- Paragon Insights listens for incoming flow data on a configured port
- Based on Paragon Insights rule configuration, data is decoded and grouped into data sets by record type (raw packet header, Ethernet frame, etc.)

- Maps the sFlow source IP (extracted from sFlow packets) to a Paragon Insights Device ID

NOTE: If no match can be made, the packet is dropped with no further decoding performed.

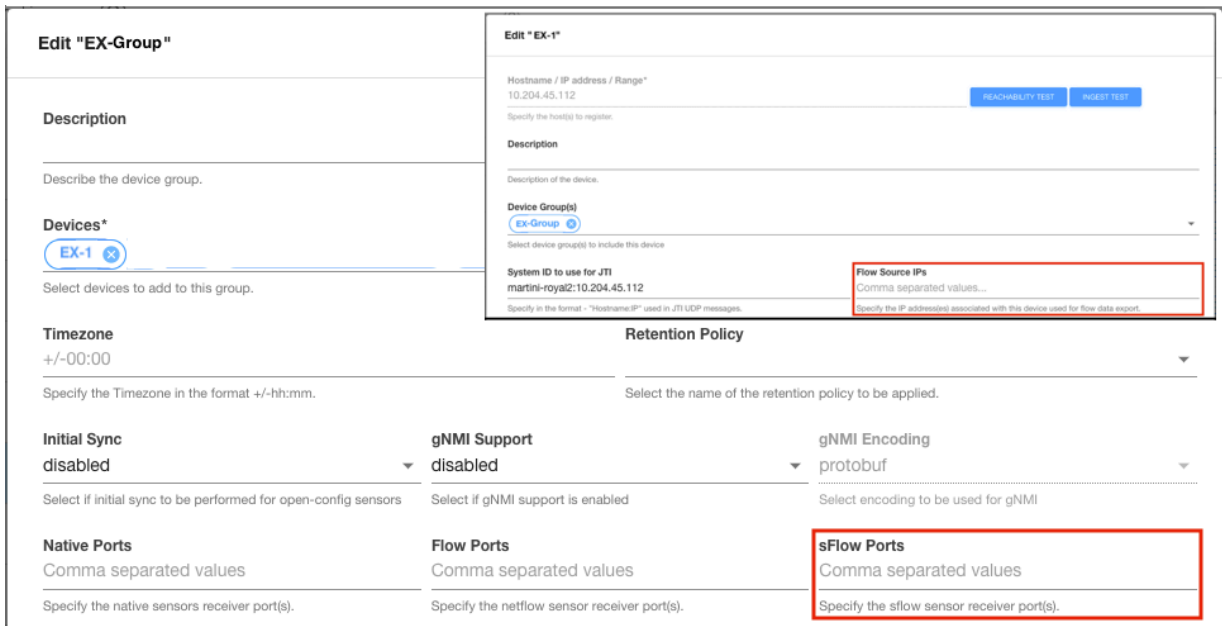
- Normalizes the fields for further use within the system
- Paragon Insights then performs further tagging, normalization, and aggregation as defined in the corresponding rule by the user.
- Finally, the time-series database, TSDB receives the data. This is where things like trigger evaluation happen.

sFlow - Configuration in Paragon Insights

As mentioned above, processing of sFlow packets depends on Paragon Insights rule configuration. It also requires that you enable sFlow in the device group or device definition. This section describes sFlow enablement, and rule and sensor configuration options for sFlow.

First, to enable sFlow, you must enter at least one IP address in the device definition under **Flow Source IPs**, and enter at least one port number in the device group definition under **sFlow Ports**. [Figure 2 on page 24](#) below is a composite image that shows the device group definition overlaid with the device definition. The appropriate sections of each window are highlighted in red.

Figure 2: Enable sFlow Composite Image



The devices in the group send their sFlow packets to Paragon Insights over the configured UDP port from the configured IP address(es). The port number(s) used in these definitions must be unique across the entire Paragon Insights installation.

NOTE:

- The **Flow Source IPs** address(es) must match an IP address that can be mapped from the **Hostname/IP Address/Range** field in the device definition. If devices send sFlow packets, but Paragon Insights cannot match the source IP to a defined device IP, then the packets are dropped without decoding.
- Paragon Insights cannot differentiate sFlow from NetFlow by looking at the packets. If you are using both NetFlow and sFlow, the port numbers must also be unique between the two flow types.

Due to the nature of sFlow and the potentially huge amount of data that can come from even a single device, we recommend the following best-practices for managing sFlow ingest:

BEST PRACTICE:

- Use unique ports from the range: UDP/49152 to UDP/65535 for sFlow.
- Use periodic aggregation to reduce the number of write procedures in the TSDB.
- Do not enable the raw table data storage option in sFlow unless sufficient high-speed storage is available for Paragon Insights TSDB.

Configure sFlow Ingest

As with other ingest methods, navigate to **Settings > Ingest** and choose the **sFlow** tab on the left of the **Ingest** window.

The **Sflow Settings** are broken down into 4 sections:

- **Sample** There are two pre-defined sample categories and each is represented in the sFlow header as an integer sample-type value. [Table 2 on page 26](#) below shows the sample types and their numeric value.

Table 2: sFlow Sample Types

Sample Type	Integer Value in sFlow Header
counter-sample	2
expanded-counter-sample	4
flow-sample	1
expanded-flow-sample	3

NOTE: The difference between the expanded sensor types and the non-expanded sample types is the size of the data fields. The field names and types are the same, but the field sizes are larger in the expanded sample types.

Packet definitions for these sample types can be found here: [sFlow Samples](#)

[Table 3 on page 26](#) shows the other fields contained in an sFlow sample header (by sample type) along with the field type.

Table 3: sFlow Packet Header Fields

field type/size in bits	counter-sample	flow-sample
integer/32	sampleSequenceNumber	sampleSequenceNumber
integer/8	sourceIDType <ul style="list-style-type: none"> • 0 = SNMP interface index • 1 = VLAN ID (smonVlanDataSource) • 2 = Physical entity (entPhysicalEntry) 	sourceIDType <ul style="list-style-type: none"> • 0 = SNMP interface index • 1 = VLAN ID (smonVlanDataSource) • 2 = Physical entity (entPhysicalEntry)

Table 3: sFlow Packet Header Fields (Continued)

field type/size in bits	counter-sample	flow-sample
integer/24	sourceIDValue	sourceIDValue
integer/32	n (the number of sampled records contained in the Counter sample)	sampleSamplingRate
integer/32	-	samplePool (number of packets that could have been sampled)
integer/32	-	sampleDroppedPackets (number of packets dropped due to lack of resources)
integer/8	-	sampleInputInterfaceFormat (input interface type)
integer/32	-	sampleInputInterfaceValue (input interface (SNMP interface index))
integer/1	-	sampleOutputInterfaceFormat (output interface type)
integer/33	-	sampleOutputInterfaceValue (SNMP interface index)
integer/32	-	n (the number of flow records)
data	counter records	flow records

- Flow Record** The **Flow Record** section provides the tools needed to define the different types of flow that might be seen in an sFlow capture. Paragon Insights ships with 16 types of pre-defined flow records, each of which have a format number and a sensor path for use in defining sFlow rules, shown in [Table 4 on page 28](#) below. There are several fields in each

type of flow record. These can be seen by selecting the desired record type from the list and clicking the **edit (pencil)** button.

Table 4: Flow Record Types

Record Type	Format Number	Sensor Path Value
raw packet headers	1	/sflow-v5/flow-sample/raw-packet-header
Ethernet frame data	2	/sflow-v5/flow-sample/ethernet-frame-data
IPv4 data	3	/sflow-v5/flow-sample/ipv4-data
IPv6 data	4	/sflow-v5/flow-sample/ipv6-data
extended switch data	1001	/sflow-v5/flow-sample/extended-switch-data
extended router data	1002	/sflow-v5/flow-sample/extended-router-data
extended gateway data	1003	/sflow-v5/flow-sample/extended-gateway-data
extended user data	1004	/sflow-v5/flow-sample/extended-user-data
extended URL data	1005	/sflow-v5/flow-sample/extended-url-data
extended MPLS data	1006	/sflow-v5/flow-sample/extended-mpls-data
extended NAT data	1007	sflow-v5/flow-sample/extended-nat-data

Table 4: Flow Record Types (Continued)

Record Type	Format Number	Sensor Path Value
extended MPLS tunnel	1008	/sflow-v5/flow-sample/extended-mpls-tunnel
extended MPLS VC	1009	/sflow-v5/flow-sample/extended-mpls-vc
extended MPLS FEC	1010	/sflow-v5/flow-sample/extended-mpls-fec
extended LVP FEC	1011	/sflow-v5/flow-sample/extended-mpls-lvp-fec
extended VLAN tunnel	1012	/sflow-v5/flow-sample/extended-vlan-tunnel

When you configure rules for sFlow, you can choose from any of these record types. You can create new flow records by clicking the **add (+)** icon on the **Sflow Settings** page.

- Counter Record** The **Counter Record** section provides the definition for the two pre-defined counter record types. There are two types of counter records, ethernet-interface-counters and generic-interface-counters. Generic interface counters are format number 1 and Ethernet interface counters are format number 2. The sensor path for generic interface counters is /sflow-v5/counter-sample/generic-interface-counter. The sensor path for Ethernet interface counters is /sflow-v5/counter-sample/ethernet-interface-counter.

The fields available within the counter records are the possible errors and the countable statistics such as:

- frame errors
- collisions
- deferred transmissions
- transmit errors
- administration status

- operational status
- input packets
- output packets
- input errors
- output errors
- and others

You can use either the generic interface counter or Ethernet interface counter in rules that you define. The counter sensors can be defined to pick even single fields from either of the available counters. You can create additional counter record types by clicking the **add (+)** icon on the **Sflow Settings** page (**Counter Record** section).

- **Protocol** The **Protocol** section provides a means to define which protocol the sFlow captures contain and allow for the decoding of many network protocols. The fields that are contained in each protocol entry are the same fields as would be seen in a frame or packet of that type. For example, an Ethernet frame would have a destination MAC address, a source MAC address, and an ethernet-next-header-type field. The fields defined in any protocol you want to decode must appear in the protocol definition in the same order as they would appear in the packet or frame.

The number column that appears is the IANA protocol number assigned to that protocol. For example, the tcp protocol is protocol number 6.

NOTE: On the **Sample**, **Flow Record**, and **Counter Record** sections, there is an **Enterprise** column. This column is for the use of vendor-specific or custom decoding details. For example, a Foundry ACL-based flow sample has the enterprise value 1991, Format 1, includes additional fields specifically for that Foundry flow. In most instances, the Enterprise value is 0.

Extend sFlow Decoding Capabilities

You can extend the protocol decoding capabilities of Paragon Insights by creating additional decoding schemas under the **Sample**, **Flow Record**, **Counter Record**, and **Protocol** tabs on the sFlow ingest settings page.

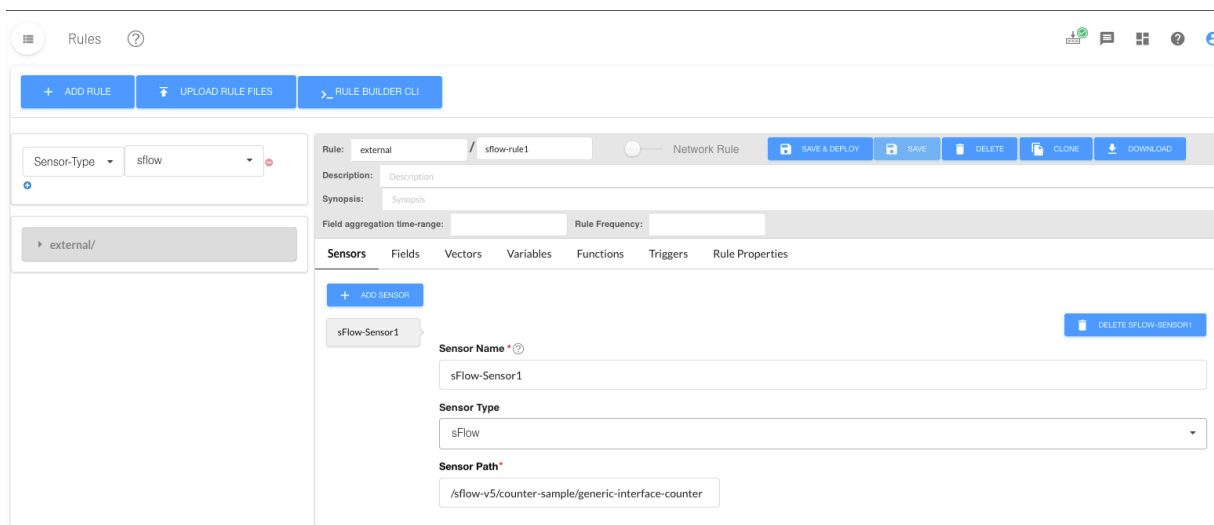
Once you have updated the ingest settings, the new rules can make use of them as follows:

- Use a packet capture utility such as Wireshark to identify the order in which the new protocols appear within the sFlow packets.
- Ensure that the sensor path you use in the sensor definition is formatted like this: `/sf1ow-v5/<sample-name>/<record-name>/<I2-protocol>/<I3-protocol>`, where *sample-name* is the new sample, *record-name* is the new counter or flow record, and the *I2-protocol* and *I3-protocol* are the new protocols. Again, these protocols must be named and positioned as they appear in the sFlow captures.

Configure sFlow Rules

As with other rule definitions, sFlow rules are made up of sensors, fields, vectors, and so on. An sFlow sensor has a **Sensor Name**, a **Sensor Type** of sFlow, and an **sFlow Path** as shown in [Figure 3 on page 31](#).

Figure 3: sFlow Sensor Definition



The sensor path serves a big role in sensor definition. Paragon Insights uses the sensor path to define not only the sFlow flow type, but the sample type, record type, protocol, and other custom path elements if needed.

Delete sFlow Settings

To delete an sFlow Setting:

1. Click **Settings** > **Ingest** from the left-nav bar.

The **Ingest Settings** page is displayed.

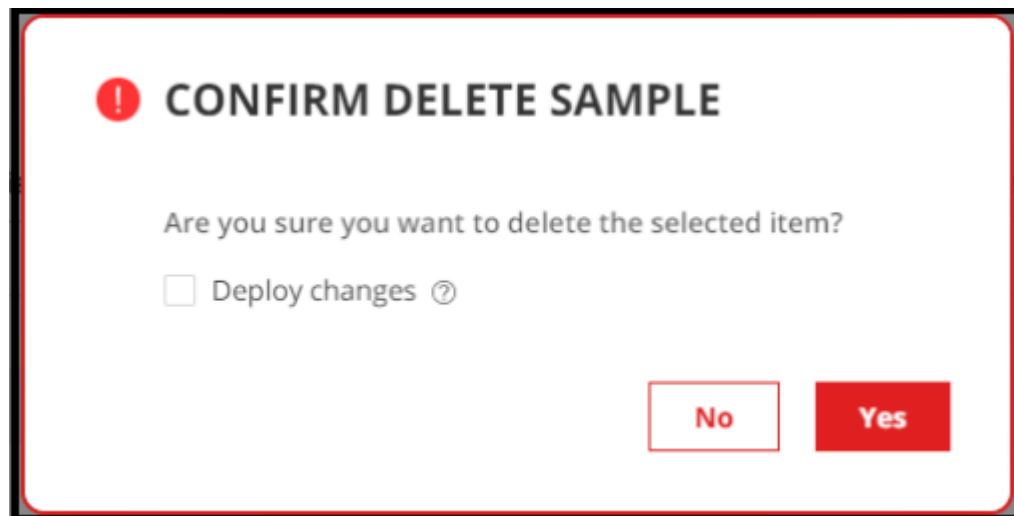
2. Click the **sFlow** tab to view the **sFlow Settings** page.

3. Do one of the following.

- To delete an sFlow sample:
 - a. Click **Sample** to view the list of sFlow samples.
 - b. Select the sFlow sample that you want to delete.
 - c. Click the **delete (trash can)** icon.

The **CONFIRM DELETE SAMPLE** pop-up appears.

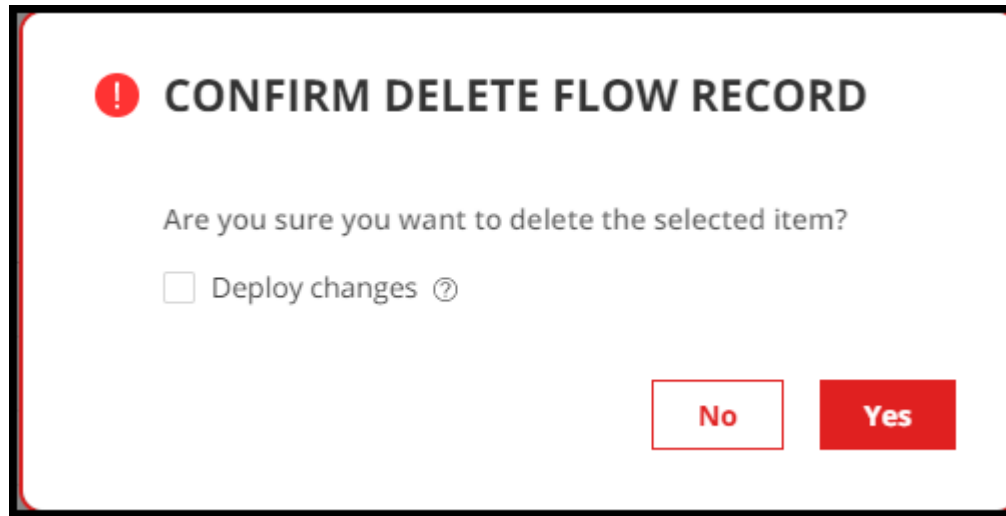
Figure 4: Confirm Delete Sample Pop-up



- To delete a flow record:
 - a. Click **Flow Record** to view the list of flow records.
 - b. Select the flow record that you want to delete.
 - c. Click the **delete (trash can)** icon.

The **CONFIRM DELETE FLOW RECORD** pop-up appears.

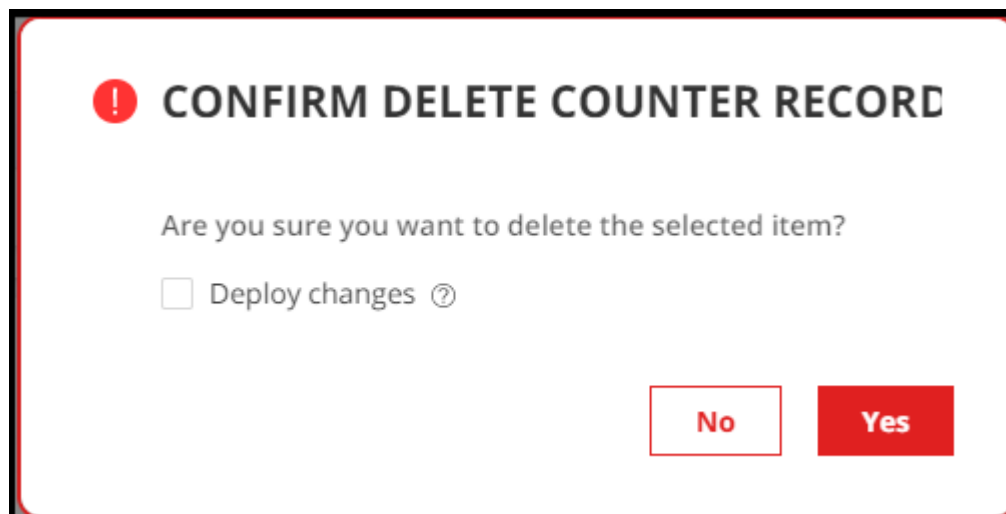
Figure 5: Confirm Delete Flow Record Pop-up



- To delete a counter record:
 - a. Click **Counter Record** to view the list of counter records.
 - b. Select the counter record that you want to delete.
 - c. Click the **delete (trash can)** icon.

The **CONFIRM DELETE COUNTER RECORD** pop-up appears.

Figure 6: Confirm Delete Pop-up

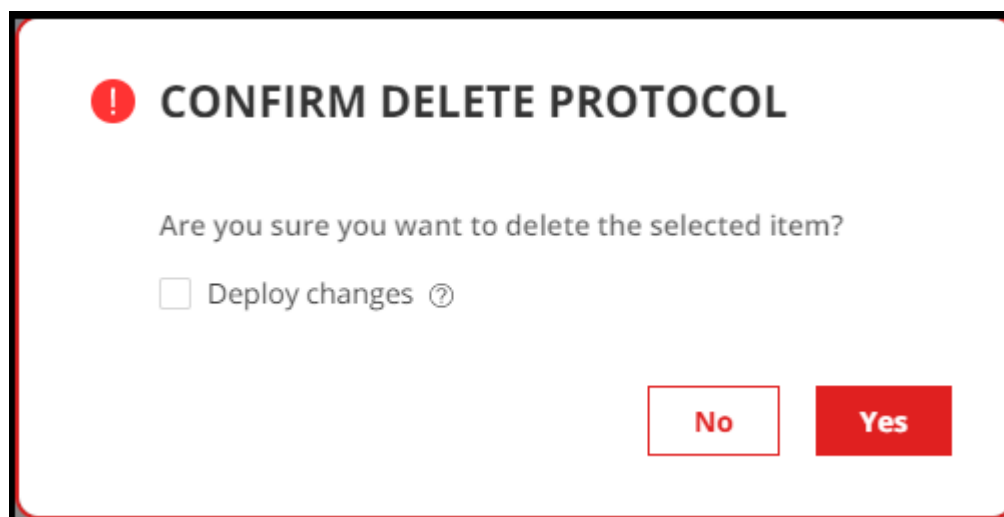


- To delete a protocol:

- a. Click **Protocol** to view the list of protocols.
- b. Select the protocol that you want to delete.
- c. Click the **delete (trash can)** icon.

The **CONFIRM DELETE PROTOCOL** pop-up appears.

Figure 7: Confirm Delete Protocol Pop-up



4. In the pop-up that appears, do any one of the following:
 - Click **Yes** to delete the sFlow setting from the database. However, the changes are not applied to the ingest service.

NOTE:

- We recommended that you do not delete an sFlow setting that is currently in use.
 - After you delete an sFlow setting from the database, you cannot configure that sFlow setting in new devices or device groups even if you have not deployed changes.
 - You can also deploy changes to the ingest service or roll back the changes that you have already deleted, from the **PENDING CONFIGURATION** page. For more information, see *Commit or Roll Back Configuration Changes in Paragon Insights*.
- Select the **Deploy changes** check box and then click **Yes** to delete the sFlow setting from the database, and to apply the changes to the ingest service.
 - (Optional) Click **No** to cancel this operation.

sFlow - Device Configuration

When you configure a device to send sFlow to a collector, you simply set an source IP address, sample-rate, polling-interval, udp-port, interface to capture from, and provide the IP address of the collector. There is no opportunity to filter or choose what data gets sent from the device side.

The following example shows the output from a switch already configured to send sFlow to a collector at IP address 10.204.32.46

```
[edit protocols sflow]
user@switch# show
polling-interval 20;
sample-rate egress 1000;
collector 10.204.32.46 {
    udp-port 5600;
}
interfaces ge-0/0/0.0;
```

OpenConfig

IN THIS SECTION

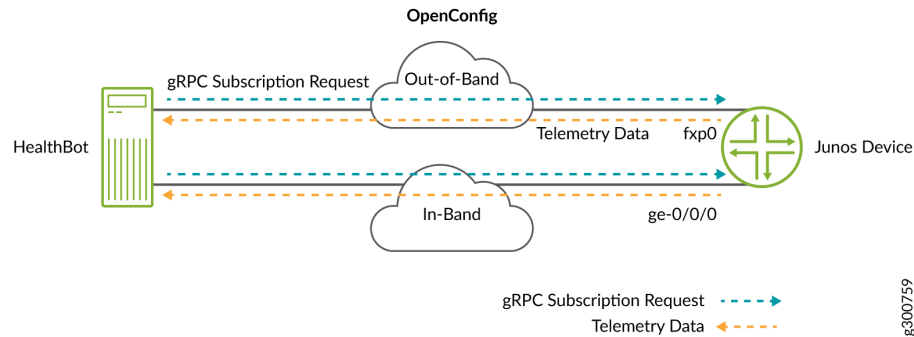
- [OpenConfig RPC | 36](#)
- [gNMI-Encoded OpenConfig RPC | 36](#)
- [OpenConfig - Device Configuration | 38](#)

This format utilizes the OpenConfig data model using gRPC. The network device acts as a gRPC server to terminate gRPC sessions from Paragon Insights, which runs as the client. RPC calls from Paragon Insights trigger the creation of sensors that either stream data periodically or report events, which are then funneled onto the appropriate gRPC channel as protobuf (GPB) messages.

In Paragon Insights releases prior to 3.1.0, OpenConfig could only be used on Junos OS and certain Cisco devices. This is because Paragon Insights has native support for the OpenConfig RPCs used by Juniper and Cisco, each of which uses its own proprietary encoding for OpenConfig data. Starting with release 3.1.0, Paragon Insights supports the gRPC Network Management Interface (gNMI) for

OpenConfig communication. This protocol allows Paragon Insights to work with many third-party OpenConfig implementations in a vendor independent way.

With OpenConfig, the device receives sensor subscriptions from Paragon Insights and pushes telemetry data through any available interface, whether in-band or out-of-band.



Note that in the figure above, the out-of-band connection is shown as connecting to the `fxp0` interface of a Junos device. If the device was from a third-party vendor, the out-of-band connection would be to the vendor-specific management port. Similarly, the in-band connection(s) would be to the vendor-specific interface designation.

OpenConfig RPC

To use OpenConfig format, you configure the device as a gRPC server. With Paragon Insights acting as the client, you define which sensors you want it to subscribe to, and it makes subscription requests towards the device.

Data streamed through gRPC is formatted in OpenConfig key/value pairs in protocol buffer (GPB) encoded messages. Keys are strings that correspond to the path of the system resources in the OpenConfig schema for the device being monitored; values correspond to integers or strings that identify the operational state of the system resource, such as interface counters. OpenConfig RPC messages can be transferred in bulk, such as providing multiple interface counters in one message, thereby increasing efficiency of the message transfer.

For more information on OpenConfig sensors, see [Understanding OpenConfig and gRPC on Junos Telemetry Interface](#).

gNMI-Encoded OpenConfig RPC

gNMI-encoded OpenConfig works much like OpenConfig RPC in that you must set the network device up as an OpenConfig server to which Paragon Insights makes subscription requests. However, gNMI supports more subscription types than Paragon Insights currently supports. Currently, Paragon Insights only supports gNMI STREAM subscriptions in the SAMPLE mode. STREAM subscriptions are long-lived

subscriptions that continue, indefinitely, to transmit updates relating to the set of paths configured within the subscription. SAMPLE mode STREAM subscriptions must include a `sample_interval`.

The messages returned to the client through gNMI are encoded by the device in protobuf, JSON, or JSON-IETF format and cannot be sent in bulk. This, in part, makes gNMI-encoded messaging less efficient than gRPC-encoded messaging.

**WARNING:**

- For JSON or JSON-IETF, it is assumed that the device returns gNMI updates as only leaf values encoded in JSON, rather than returning an entire hierarchy or sub-hierarchy as a JSON object.
- Numbers encoded in JSON or JSON-IETF are decoded by Paragon Insights as either float64, int64, or string, according to [RFC 7159](#) and [RFC 7951](#). If your OpenConfig rules contain fields that are of a different type, we recommend that you change the field types accordingly.

Junos OS and Cisco devices can be managed by Paragon Insights using gNMI-encoded OpenConfig. If a device does not support gNMI in general, or the STREAM subscription in SAMPLE mode, or does not support an OpenConfig request, it returns one of the following errors:

- Unimplemented
- Unavailable
- InvalidArgument

In the case of a Junos OS or Cisco device, the error causes the connection to fall back to OpenConfig RPC. In the case of a third-party device, the connection simply fails due to the error.

gNMI-encoded OpenConfig can be enabled at the device or device-group level. If enabled at the device-group level, then all devices added to the group use gNMI by default. If enabled (or not enabled) at the device level, then the device level setting takes precedence over the device-group level setting.

NOTE: During the initial connection gNMI devices attempt to perform an initial sync with the client. The device sends a continuous stream of data until the device and the collector (Paragon Insights) are in sync. After initial sync, the device begins normal streaming operations based on the configured reporting rate. Because of the processing load this creates, Paragon Insights has this feature disabled by default. It can be enabled at the device-group or device level if needed.

For more information about gNMI, see: [gRPC Network Management Interface \(gNMI\)](#).

OpenConfig - Device Configuration

OpenConfig requires:

- Junos OS Version: 16.1 or later
- The OpenConfig sensor requires that the Junos device have the OpenConfig and network agent packages installed. These packages are built into Junos OS Releases 18.2X75, 18.3, and later. For releases between 16.1 and 18.2X75 or 18.2, you must install the packages.

For information on installing network agent package, see [Installing the Network Agent Package \(Junos Telemetry Interface\)](#) topic.

To verify whether you have these packages, enter the following command:

```
user@host> show version | match "Junos:|openconfig|na telemetry"

Junos: 19.2R1.8
JUNOS na telemetry [19.2R1.8]
JUNOS Openconfig [19.2R1.8]
```

See [Understanding OpenConfig and gRPC on Junos Telemetry Interface](#) for more information.

- Network agent is not supported on PPC platforms (MX104, MX80, and so on)
- Required configuration:

```
set system services extension-service request-response grpc clear-text port number
```

Syslog

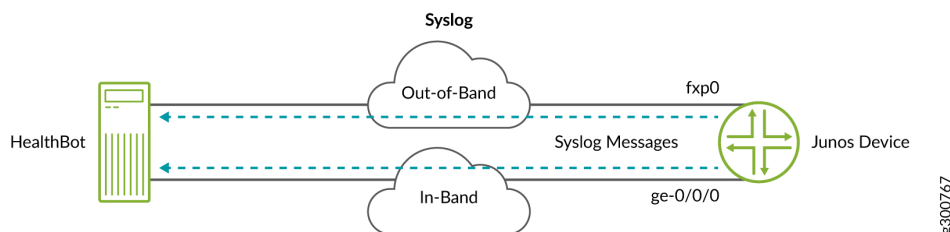
IN THIS SECTION

- [Overview | 39](#)
- [System-Generated Fields | 40](#)
- [Usage Notes | 41](#)
- [Optional Configuration Elements | 41](#)

- [Example: Creating a Rule Using Syslog Ingest](#) | 43
- [CONFIGURE NETWORK DEVICES](#) | 44
- [SET UP SYSLOG INGEST](#) | 44
- [CREATE RULE, APPLY PLAYBOOK](#) | 52

In addition to the JTI-related options above, starting with Release 2.1.0, Paragon Insights also supports syslog natively as another data collection method, using a data model that aligns with other Paragon Insights ingest mechanisms to provide all the same feature richness.

A device can push syslog messages (when configured) over UDP to the Paragon Insights server either out-of-band through the Routing Engine (RE) using the router's management interface, or in-band through the Packet Forwarding Engine, that is, directly from a line card.



To use syslog format, you configure the device with settings that include where to send the syslog messages. See "[Syslog - Device Configuration](#)" on [page 43](#) below for details. When you configure Paragon Insights to start collecting the data, messages are already flowing towards the server.

For more information on syslog as used by Juniper devices, see [Junos OS System Log Overview](#).

Overview

Syslog ingest requires some setup before you can use it as a sensor in a rule:

- **Pattern** - A pattern identifies some syslog event; you create a pattern for each event you want to monitor. You can configure patterns for both structured and unstructured events.
- **Pattern set** - With the patterns configured, you then group them into a pattern set, which you then reference when defining the syslog sensor settings within a rule.

To see how patterns and pattern sets are used, see "[Example: Creating a Rule Using Syslog Ingest](#)" on [page 43](#).

System-Generated Fields

Some fields are common in syslog messages. Paragon Insights extracts these fields and includes them automatically in the raw table, enabling you to make use of them directly when creating a rule, and avoiding the need to configure patterns.

To illustrate use of these values, consider the following example syslog messages:

Structured - <30>1 2019-11-22T03:17:53.605-08:00 R1 mib2d 28633 SNMP_TRAP_LINK_DOWN [junos@2636.1.1.1.2.29 snmp-interface-index="545" admin-status="up(1)" operational-status="down(2)" interface-name="ge-1/0/0.16"] ifIndex 545, ifAdminStatus up(1), ifOperStatus down(2), ifName ge-1/0/0.16

Equivalent unstructured - <30>Nov 22 03:17:53 R1 mib2d[28633]: SNMP_TRAP_LINK_DOWN: ifIndex 545, ifAdminStatus up(1), ifOperStatus down(2), ifName ge-1/0/0.16

System-generated fields:

- "__log_priority__" - Priority of syslog message
 - In the examples, <30> denotes the priority
- "__log_timestamp__" - Timestamp in epoch in the syslog message
 - In the structured example, 2019-11-22T03:17:53.605-08:00 is converted to epoch with **-08:00** indicating the time zone
 - In the unstructured example, the time zone from the configuration will be used to calculate epoch
- "__log_host__" - Host name in the syslog message
 - In the examples, R1 denotes the host name
- "__log_application_name__" - Application name in the syslog message
 - In the examples, mib2d is the application name
- "__log_application_process_id__" - Application process ID in the syslog message
 - In the examples, 28633 is the ID
- "__log_message_payload__" - Payload in the message
 - Structured example - "SNMP_TRAP_LINK_DOWN [junos@2636.1.1.1.2.29 snmp-interface-index="545" admin-status="up(1)" operational-status="down(2)" interface-name="ge-1/0/0.16"] ifIndex 545, ifAdminStatus up(1), ifOperStatus down(2), ifName ge-1/0/0.16"
 - Unstructured example - "SNMP_TRAP_LINK_DOWN: ifIndex 545, ifAdminStatus up(1), ifOperStatus down(2), ifName ge-1/0/0.16"

- "Event-id" - Denotes the event ID configured in the pattern
 - In the examples, SNMP_TRAP_LINK_DOWN is the event ID

NOTE: Be sure not to define any new fields using a name already defined above.

Usage Notes

- If you add a device in Paragon Insights using its IP address (and the address is not the actual host name of the device), you must also add the device's host name in the **Syslog Hostnames** field.
- Multiple device groups can listen for syslog messages using the same port.
- A configured time zone is not considered when processing structured messages, as the messages themselves include the time zone.
- Daylight savings time is not currently supported.

Optional Configuration Elements

Configure Syslog Ports

By default, Paragon Insights listens for syslog messages from all device groups on UDP port 514. You can change the system-level syslog port, as well as configure one or more ports per device group. The more specific device group setting takes precedence over the system level setting.

To change the system-level syslog port:

1. Click **Settings > Ingest** in the left-nav bar.
2. Select the **Syslog** tab on the Ingest Settings page.
3. On the Syslog page, edit the port number.
4. Click **Save & Deploy**.

To configure a syslog port for a device group:

1. Go to the **Configuration > Device Group** page and click on the name of a device group.
2. Click the **Edit (pencil)** icon.
3. In the pop-up window, enter the port(s) in the **Syslog Ports** field.
4. Click **Save & Deploy**.

Configure Time Zone for a Device

When a device exports structured syslog messages, time zone information is included within the message. However, unstructured syslog messages do not include time zone information. By default, Paragon Insights uses GMT as the time zone for a device. In these cases you can assign a time zone to a device or device group within Paragon Insights.

To configure a device's time zone at the device group level:

1. Go to the **Configuration > Device Group** page and click on the name of a device group.
2. Click the **Edit (pencil)** icon.
3. In the pop-up window, enter a value in the **Timezone** field, for example **-05:00**.

To configure a device's time zone at the device level:

1. Go to the **Configuration > Device** page and click on the name of a device.
2. Click the **Edit (pencil)** icon.
3. In the pop-up window, enter a value in the **Timezone** field, for example **-05:00**.

NOTE: The more specific device setting takes precedence over the device group setting.

Configure Multiple Source IP Addresses for a Device

In cases where syslog messages arrive from a device using a different source IP address than the one originally configured in the Paragon Insights GUI, you can add additional source IP addresses.

To support additional source IP addresses:

1. Go to the **Configuration > Device** page and click on the name of a device.
2. Click the **Edit (pencil)** icon.
3. In the pop-up window, enter the IP address(es) in the **Syslog Source IPs** field.

Configure Host Name Aliases for a Device

When a device has more than one host name, such as a device with dual REs, syslog messages can arrive at the Paragon Insights server with a host name that is not the device's main host name. In these cases, you can add host name aliases for that device.

NOTE: If you add a device in Paragon Insights using its IP address, you must also add the host name that will appear in the syslog messages.

To configure additional hostname aliases:

1. Go to the **Configuration > Device** page and click on the name of a device.
2. Click the **Edit (pencil)** icon.
3. In the pop-up window, enter the host name(s) in the **Syslog Host Names** field.

Syslog - Device Configuration

Syslog requires that you have at least the following configuration elements committed on your device(s) in addition to

- Junos OS Version: Any release
- Required configuration:

```
set system syslog host 110.1x.x0.1 any any
set system syslog host 10.1x.x0.1 allow-duplicates
set system syslog host 10.1x.x0.1 structured-data
```

10.1x.x0.1 = Paragon Insights server

Example: Creating a Rule Using Syslog Ingest

To illustrate how to configure and use a syslog sensor, consider a scenario where you want to:

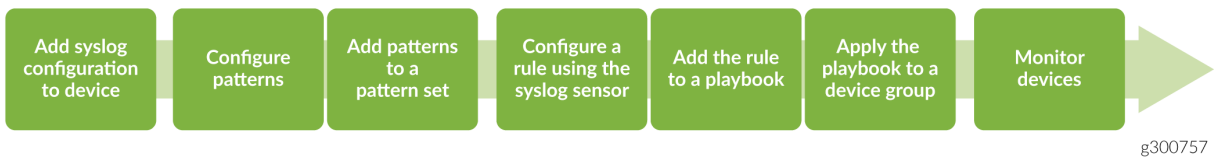
- Monitor interface operational down status
- Use two syslog events, one structured and one unstructured
 - [SNMP_TRAP_LINK_DOWN](#)
 - PSEUDO_FPC_DOWN
- Use a rule with a trigger to indicate when the interface goes down

To implement this scenario, you will need to complete the following activities:

- ["CONFIGURE NETWORK DEVICES" on page 44](#)

- ["SET UP SYSLOG INGEST" on page 44](#)
- ["CREATE RULE, APPLY PLAYBOOK" on page 52](#)
- ["Monitor the Devices" on page 59](#)

The workflow is as follows:



CONFIGURE NETWORK DEVICES

NOTE: This example assumes you have already added your devices into Paragon Insights and assigned them to a device group.

Add Syslog Configuration to the Network Device

If not already done, configure your network device(s) to send syslog data to Paragon Insights. The example device configuration from the previous section is repeated here:

```

set system syslog host 10.1x.x0.1 any any
set system syslog host 10.1x.x0.1 allow-duplicates
set system syslog host 10.1x.x0.1 structured-data

```

10.1x.x0.1 = Paragon Insights server

SET UP SYSLOG INGEST

Configure Events Patterns

An Event pattern is a configuration to monitor some syslog event; you create a pattern for each event you want to monitor. This example uses patterns to monitor four syslog events (two structured and two unstructured).

NOTE: See the usage notes at the end of this section for more detail on what has been configured.

1. In the Paragon Insights GUI, click **Settings > Ingest** in the left-nav bar.
2. Select the **Syslog** tab on the Ingest Settings page.
3. On the Syslog page, click the plus (+) icon.
4. In the pop-up window that appears, enter the following parameters for the first pattern, named **snmp-if-link-down**:

ADD PATTERN ✕

Name * ✔

Event Id *

Description

Filter

Field + ✎ 🗑️

	Name*	Description	From*	Type*	Key Field
<input type="radio"/>	if-name		interface-name	integer	No
<input checked="" type="radio"/>	<input style="width: 80px;" type="text" value="snmp-index"/>	<input style="width: 80px;" type="text"/>	<input style="width: 80px;" type="text" value="snmp-interface-index"/>	<input style="width: 80px;" type="text" value="integer"/> ▾	<input type="checkbox"/> No

2 items ✓ ✕

Constant + ✎ 🗑️

Cancel
Save
Save And Deploy

5. Click **Save and Deploy**.
6. Click the plus (+) icon once more and enter the following parameters for the second pattern, named **fpc-offline**:

ADD PATTERN ✕

Name * ✔

Event Id *

Description

Filter

Field + ✎ 🗑

Name*	Description	From*	Type*	Key Field

NOTE: The full value entered in the Filter field is **fpc%(NUMBER:fpc) Marking ports % {WORD:port-status}**

7. Click **Save and Deploy**. On the Syslog page you should see the two patterns you just created.

Usage notes for the patterns

For structured syslog:

- The event ID (**SNMP_TRAP_LINK_DOWN**) references the event name found within the syslog messages.
- Fields are optional for structured syslog messages; if you don't configure fields, the attribute names from the message will be treated as field names.
 - In this example, however we have user-defined fields:
 - The field names (**if-name**, **snmp-index**) are user-defined.
 - The field **interface-name** value is an attribute from the syslog message, for example, `ge-0/3/1.0`; this field is renamed as **if-name**
 - The field **snmp-interface-index** value is an attribute from the syslog message, for example, `ifIndex 539`; this field is renamed as **snmp-index**
 - The field **snmp-interface-index** here is defined as an integer; by default the fields extracted from a syslog message are of type string, however type integer changes this to treat the value as an integer
- The constant section is optional, in this example, we have user-defined constants.
 - The constant name **ifOperStatus** is user-defined; in this case it has the integer value of '2'

- Filter configuration is optional for a structured syslog, though you can do so if desired; if used, the filter-generated fields will override the fields included in the syslog message.
- The key fields section is optional; by default the hostname and event ID will be the keys used by Paragon Insights; add additional key fields here; in this example, we have **key-fields**, namely **interface-name**, where the name and value are extracted from the syslog message's attribute-value pair

For unstructured syslog:

- The event ID is user defined, this case **PSEUDO_FPC_DOWN**
 - For example, neither the unstructured syslog `Nov 22 02:27:05 R1 fpc1 Marking ports down` nor its structured counterpart `<166>1 2019-11-22T02:38:23.132-08:00 R1 - - - fpc1 Marking ports down` includes an event ID
- A filter must be used to derive fields (unlike proper structured syslog); this example uses `fpc%{NUMBER:fpc} Marking ports %{WORD:port-status}`, where **fpc** becomes the field name and **NUMBER** denotes the syntax used to extract the characters out of that particular portion of the message, for example "2"
 - An example of a syslog message that matches the grok filter is "fpc2 Marking ports down"
- constant **fpc-status** - has a string value of 'online'

Regarding filters:

- By default in a pattern, field and constant values are a string; to treat it as an integer or float, define the pattern's field type as integer or float
- For unstructured patterns, you must configure a filter as the messages are sent essentially as plain text and don't include field info on their own
- Filters should always be written to match the portion of message after the event ID; this allows the filter to parse a syslog message irrespective of whether it arrives in unstructured or structured format
 - For example, the filter `fpc%{NUMBER:fpc} Marking ports %{WORD:port-status}` matches both versions of the following syslog message:
 - Structured: `<166>1 2019-11-22T02:38:23.132-08:00 R1 - - - fpc1 Marking ports down`
 - Unstructured: `Nov 22 02:27:05 R1 fpc1 Marking ports down`

Clone an Existing Syslog Events Pattern

Starting with Paragon Insights Release 4.0.0, you can clone an existing Syslog pattern.

To clone an existing Syslog events pattern:

1. Click **Settings > Ingest** in the left-nav bar.

The **Ingest Settings** page is displayed.

2. Click the **Syslog** tab to view the **Syslog** page.
3. Click to expand the **Events Pattern** section in the **Syslog Settings** page to view existing syslog events patterns.
4. To clone a pattern for Paragon Insights Release 4.0.0 and 4.1.0, click the **Clone** icon.

To clone a particular template for Paragon Insights Release 4.2.0 and later, select the option button next to the name of the template and click **Clone**.

The **Clone Pattern: <name of syslog pattern>** page is displayed.

From the Clone Pattern: <name of syslog pattern> page, you can

- Edit existing fields
 - Add new fields or constants
 - Add or remove key fields
5. Click **Save** to save configuration and clone the syslog pattern.

Alternatively, click **Save & Deploy** to save configuration, clone syslog pattern, and deploy the pattern.

Add Patterns to a Pattern Set

With the patterns configured, group them into a pattern set.

1. On the Syslog page, click to expand the **Pattern Set** section and click the plus (+) icon.
2. In the pop-up window that appears, enter the following parameters:

ADD PATTERN-SET

Name * ✓

Description

Patterns *

3. Click **Save and Deploy**. On the Syslog page you should see the pattern set you just created.

Clone an Existing Syslog Pattern Set

Starting with Paragon Insights Release 4.0.0, you can clone an existing Syslog pattern set.

To clone an existing Syslog pattern set:

1. Click **Settings > Ingest** in the left-nav bar.

The **Ingest Settings** page is displayed.

2. Click the **Syslog** tab to view the **Syslog** page.

3. Click to expand the **Pattern Set** section in the **Syslog** page to view existing syslog pattern sets.

4. To clone a pattern set for Paragon Insights Release 4.0.0 and 4.1.0, click the **Clone** icon.

To clone a particular template for Paragon Insights Release 4.2.0 and later, select the option button next to the name of the template and click **Clone**.

The **Clone Pattern-set: <name of pattern-set>** page is displayed.

From the **Clone Pattern-set: <name of pattern-set>** page, you can

- Edit the name and description fields
- Add or remove patterns from the Patterns field.

5. Click **Save** to save configuration and clone the syslog pattern set.

Alternatively, click **Save & Deploy** to save configuration, clone syslog pattern set, and deploy the pattern set.

Configure Header Pattern

Starting in Paragon Insights Release 4.0.0, you can configure the pattern for parsing the header portion of a syslog message. With this release, unstructured syslog messages of non-Juniper devices are supported. In earlier releases, you can only parse the payload portion of either a structured syslog message as specified in [RFC 5424](#) standard, or a Juniper device's unstructured syslog message.

In general, it is assumed that any unstructured syslog message matches the Juniper syslog message pattern. For example, you do not have to configure a Juniper header pattern as this pattern is inbuilt with Paragon Insights. However, in case of a non-Juniper device's unstructured syslog message that does not match with the inbuilt pattern, a first match is made with one of the user-configured header patterns. Following a successful match, the fields are extracted. When there is no match, the incoming syslog message is dropped.

To configure a header pattern:

1. Navigate to **Settings > Ingest** in the left-nav bar.

The **Ingest Settings** page is displayed.

2. Click **Syslog** to view the **Syslog** page.
3. Click to expand the **Header Pattern** section.

The **Header Pattern** section of the **Syslog** page is displayed. You can add a new header pattern and edit or delete an existing header pattern from this page.

4. Click the plus (+) icon to add a new header.

The **Add Header Pattern** page is displayed.

5. Enter the following information in the **Add Header Pattern** page.

- a. Enter a name for the header pattern in the **Name** field.
- b. Enter a description for the header pattern in the **Description** field.

For example, you can provide a one-line description of why you are creating this header pattern.

- c. Enter the filter or regular expression (regex) for the header pattern in the **Filter** field.

NOTE: You can use regex101.com to edit, validate, and modify the filter pattern you want to add to the header pattern.

An example of a filter pattern is `(.*):([A-Z][a-z]{2} \d{1,2} \d{1,2}:\d{1,2}:\d{1,2}\.d*)\s:\s([a-z]*)\s([\d*])\s*(.*)\s*`.

- d. **log-host**, **log-timestamp**, and **log-message-payload** of the Fields section are mandatory fields that determine the position of the header.

In the Fields section,

- i. Click **log-host**, and enter the following information.
 - Enter a name for the log host in the **Name** field.
log-host is the default name.
 - Enter a description for log-host in the **Description** field.
The default description is **Position of host name**.
 - Enter the capture group value with prefix **\$** in the **From** field.

The capture group determines from which position in the header the **log-host** starts.

ii. Click **log-timestamp**, and enter the following information.

- Enter a name for the log timestamp in the **Name** field.

log-timestamp is the default name.

- Enter a description for log-timestamp in the **Description** field.

The default description is **Position of time stamp**.

- Enter the capture group value with prefix **\$** in the **From** field.

The capture group determines from which position in the header the **log-timestamp** starts.

NOTE: Ensure that timestamp format follows this sample timestamp format: "Jan _2 15:04:05 2006". Otherwise parsing of syslog messages will lead to an undefined behaviour.

iii. Click **log-message-payload**, and enter the following information.

- Enter a name for the log message payload in the **Name** field.

log-message-payload is the default name.

- Enter a description for log-message-payload in the **Description** field.

The default description is **Position of payload**.

- Enter the capture group value with prefix **\$** in the **From** field.

The capture group determines from which position in the header the **log-message-payload** starts.

iv. (Optional) Click the plus (+) icon to add a new field.

- Enter a name for the new field in the **Name** field.
- Enter a description for the new field in the **Description** field.
- Enter the capture group value with prefix **\$** in the **From** field.

The capture group determines from which position in the header the new field starts.

You can add one or more than one fields by clicking the plus (+) icon.

6. Enter the name(s) of key fields in the **Key Fields** text box.

7. Click **Save** to save configuration, or click **Save & Deploy** to save and immediately deploy the configuration.

Alternatively, to cancel the configuration, click **Cancel**.

Editing an Existing Header Pattern

To edit an already configured header pattern:

1. Navigate to **Settings > Ingest** in the left-nav bar.

The **Ingest Settings** page is displayed.

2. Click **Syslog** to view the **Syslog** page.

3. Click to expand the **Header Pattern** section.

The Header Pattern section of the **Syslog Setting** page is displayed.

4. Select the header pattern you want to edit by selecting the check box next to the name of the header pattern, and click the **Edit (pencil)** icon.

The Edit Header Pattern *Header Name* page is displayed.

5. After you have edited the required fields, click **Save** to save configuration.

You can also click **Save & Deploy** to save and immediately deploy the edited configuration.

CREATE RULE, APPLY PLAYBOOK

Configure a Rule Using the Syslog Sensor

With the syslog ingest settings complete, you can now create a rule using syslog as the sensor.

This rule includes three elements:

- A syslog sensor
- Four fields capturing data of interest
- A trigger that indicates when the interface goes down

NOTE: See the usage notes at the end of this section for more detail on what has been configured.

1. Click **Configuration > Rules** in the left-nav bar.

2. On the Rules page, click the **+ Add Rule** button.
3. On the page that appears, in the top row of the rule window, set the rule name. In this example, it is **check-interface-status**.
4. Add a description and synopsis if you wish.
5. Click the **+ Add sensor** button and enter the following parameters in the Sensors tab:

The screenshot shows the 'Sensors' configuration interface. At the top, there are tabs for 'Sensors', 'Fields', 'Vectors', 'Variables', 'Functions', 'Triggers', and 'Rule Properties'. The 'Sensors' tab is selected. On the left, there is a list of sensors with 'if-status-sensor' highlighted. A '+ Add sensor' button is at the top left, and a 'Delete if-status-sensor' button is at the top right. The main form has the following fields: 'Sensor Name' (if-status-sensor), 'Sensor type' (Syslog), 'Pattern set' (check-interface-status), and 'Maximum hold period' (Enter syslog Maximum hold period).

6. Now move to the Fields tab, click the **+ Add field** button, and enter the following parameters to configure the first field, named **event-id**:

The screenshot shows the 'Fields' configuration interface. At the top, there are tabs for 'Sensors', 'Fields', 'Vectors', 'Variables', 'Functions', 'Triggers', and 'Rule Properties'. The 'Fields' tab is selected. On the left, there is a list of fields with 'event-id' highlighted. A '+ Add field' button is at the top left, and a 'Delete event-id' button is at the top right. The main form has the following fields: 'Field name' (event-id), 'Description' (Add a description for this field), 'Field type' (Field type), 'Add to rule key' (unchecked), 'Ingest type (Field source)' (Sensor), 'Sensor' (if-status-sensor), 'Path' (event-id), and 'Data if missing' (Zero suppression, Default value).

7. Click the **+ Add field** button again and enter the following parameters to configure the second field, named **fpc-slot**:

The screenshot shows the 'Fields' configuration page with the following settings for the 'fpc-slot' field:

- Field name:** fpc-slot
- Description:** Add a description for this field
- Field type:** Field type
- Add to rule key:**
- Ingest type (Field source):** Sensor
- Sensor:** if-status-sensor
- Path:** fpc
- Zero suppression:**
- Data if missing:** Default value

- 8. Click the **+ Add field** button again and enter the following parameters to configure the third field, named **if-name**:

The screenshot shows the 'Fields' configuration page with the following settings for the 'if-name' field:

- Field name:** if-name
- Description:** Add a description for this field
- Field type:** Field type
- Add to rule key:**
- Ingest type (Field source):** Sensor
- Sensor:** if-status-sensor
- Path:** if-name
- Zero suppression:**
- Data if missing:** all interfaces

- 9. Click the **+ Add field** button once more and enter the following parameters to configure the fourth field, named **snmp-index**:

Sensors **Fields** Vectors Variables Functions Triggers Rule Properties

+ Add field

event-id
fpc-slot
if-name
snmp-index

Field name*

snmp-index

Description

Add a description for this field

Field type

Field type

Add to rule key

Ingest type (Field source)

Sensor

Sensor **Path*** **Data if missing**

if-status-sensor snmp-index Zero suppression Default value

10. Now move to the Triggers tab, click the **+ Add trigger** button, and enter the following parameters to configure a trigger named **link-down**:

The screenshot displays the 'Triggers' configuration page in a web interface. At the top, there are navigation tabs: Sensors, Fields, Vectors, Variables, Functions, Triggers (selected), and Rule Properties. Below the tabs, there is a '+ ADD TRIGGER' button and a 'link-down' trigger card with a 'DELETE LINK-DOWN' button. The main configuration area shows two trigger rules. The first rule, 'is-link-down', has a Trigger Name of 'link-down' and a Frequency of '2s'. It includes a 'Disable alarm deduplication' toggle. The 'WHEN' section is configured with 'Left operand' as '\$event-id', 'Operator' as '=~', and 'Right operand' as 'SNMP_TRAP_LINK-DOWN'. It also has an 'All in time range' of '300s' and an 'Ignore Case' toggle. The 'THEN' section has a 'Color' set to red and a 'Message' that reads 'Link down for \$if-name(\$snmp-id: \$snmp-index)'. The second rule, 'is-fpc-down', has a Trigger Name of 'is-fpc-down' and a Frequency of '300s'. Its 'WHEN' section has 'Left operand' as '\$event-id', 'Operator' as '=~', and 'Right operand' as 'PSEUDO_FPC_DOWN'. The 'THEN' section has a 'Color' set to red and a 'Message' that reads 'Link down for \$if-name of FPC\$[fpc:slot]'. A note at the bottom of the configuration area states: 'Functions can be used as Trigger actions too, define them using the 'Functions' menu at the top.'

11. At the upper right of the window, click the + Save & Deploy button.

Usage Notes for the rule

- Sensor tab

- The sensor name **if-status-sensor** is user-defined
- The sensor type is **syslog**
- Pattern set **check-interface-status** - reference to the pattern set configured earlier
- If not set, the Maximum hold period defaults to 1s
- Fields tab
 - Four fields are defined; although the patterns are capturing more than four fields of data, this example defines four fields of interest here; these fields are used in the trigger settings
 - The field names (**event-id**, **fpc-slot**, **if-name**, **snmp-index**) are user-defined
 - path **event-id** - default field created by syslog ingest in the raw table; references the field from the pattern configuration
 - path **fpc** - references the value from the filter used in the unstructured pattern configuration
 - path **if-name** - references the field from the pattern configuration
 - Data if missing **all interfaces** - if the if-name value is not included in the syslog message, use the string value "all interfaces"
 - path **snmp-index** - references the field from the pattern configuration
- Triggers tab
 - The trigger name **link-down** is user-defined
 - **frequency 2s** - Paragon Insights checks for link-down syslog messages every 2 seconds
 - term **is-link-down** - when **\$event-id** is like **SNMP_TRAP_LINK_DOWN**, in any syslog message in the last **300 seconds**, make **red** and show the message **Link down for \$if-name(snmp-id: \$snmp-index)**
 - **\$event-id** - \$ indicates to reference the rule field **event-id**
 - **Link down for \$if-name(snmp-id: \$snmp-index)** - for example, "Link down for ge-2/0/0 of FPC 2"
 - **\$if-name** - references the field value, i.e., the name of the interface in the syslog message
 - term **is-fpc-down** - when **\$event-id** is like **PSEUDO_FPC_DOWN**, in any syslog message in the last 300 seconds, make **red** and show the message **Link down for \$if-name of FPC\$fpc-slot**
 - **\$event-id** - \$ indicates to reference the rule field **event-id**
 - **\$if-name** - "all interfaces"

- Link down for \$if-name of FPC\$fpc-slot - for example, "Link down for all interfaces of FPC 2"

Add the Rule to a Playbook

With the rule created, you can now add it to a playbook. For this example, create a new playbook to hold the new rule.

1. Click **Configuration > Playbooks** in the left-nav bar.
2. On the Playbooks page, click the **+ Create Playbook** button.
3. On the page that appears, enter the following parameters:

Create Playbook

Name* ?

Synopsis ?

Detailed description about the playbook and usage details

Rules* ?

Cancel Save Save & Deploy

4. Click **Save & Deploy**.

Apply the Playbook to a Device Group

To make use of the playbook, apply it to a device group.

1. On the Playbooks page, click the **Apply (Airplane)** icon for the **check-interface-status** playbook.
2. On the page that appears:
 - Enter a name
 - Select the desired device group
 - Click **Run Instance**

3. On the Playbooks page, confirm that the playbook instance is running. Note that the playbook may take some time to activate.

Monitor the Devices

With the playbook applied, you can begin to monitor the devices.

1. Click **Monitor > Device Group Health** in the left-nav bar, and select the device group to which you applied the playbook from the **Device Group** pull-down menu.
2. Select one or more of the devices to monitor.
3. In the Tile View, the tile labeled external contains the parameters from the rule you configured earlier.

NOTE: For this example, since the rule trigger does not include a 'green' term, the status will show red when there is an issue and otherwise will show gray (no data).

SNMP Trap and Inform Notifications

IN THIS SECTION

- [Glossary | 60](#)
- [Configurations | 62](#)

Paragon Insights Release 4.0.0 supports inform and trap notifications that are sent by devices in the network for fault management. Traps and informs are notifications about change of state in network that are sent between the *SNMP manager* (Paragon Insights) and the *SNMP agents* (devices), on which Paragon Insights performs trigger evaluations. Paragon Insights processes traps and informs from the configured device only if a playbook containing an SNMP-notification rule is running for the specified device. In all other cases, the trap or inform message is dropped by the SNMP Manager.

The following sections describe relevant terms, configuration of traps and informs through CLI, port configuration, and accessing status of SNMP traps through CLI.

NOTE: SNMP trap notifications are supported by SNMPv2c and SNMPv3. SNMP inform messages are supported only when you use SNMPv3 protocol.

Glossary

The following terms are used when describing processes or concepts related to SNMP traps and informs.

- *Authoritative agent* – In SNMPv3 transactions between two entities (agent and manager), the flow of sending notification is controlled through authentication and privacy that are unique features in SNMPv3. Authentication identifies and verifies the source of an SNMPv3 message. The privacy feature prevents packet analyzers from snooping the content of messages by encrypting them.

The entity that controls the notification flow is known as authoritative agent. In SNMPv3, the non-authoritative entity must know the *<Engine ID>* of the authoritative agent for a successful communication.

- *Traps or trap messages* – A trap is an unacknowledged notification sent from an SNMP agent to the SNMP manager. In trap messages, SNMP agent is the authoritative agent. The administrator must configure the SNMP v3 *<user>* (distinct from the local IAM users) and *<Context Engine ID>* on the device that sends out the trap messages. For traps, the *<Context Engine ID>* is set to the *Engine ID* that uniquely identifies the SNMP agent.
- *Informs or inform messages* – An inform is also a notification sent from an SNMP agent to the SNMP manager. In inform messages, SNMP manager is the authoritative agent. The configuration is done on the device that needs to send inform messages, with the details of the remote authoritative agent, SNMP manager. The administrator must configure the *<user>* found in the remote SNMP manager.
- *Engine ID* – *<Engine ID>* is a hexadecimal generated for a given agent that uniquely identifies the SNMP agent and needs to be unique across a given administrative domain. It also must be persistent across reboots or upgrades.
- *Security Engine ID* – It is a security parameter in the SNMP communication between the agent and the manager. It is usually set to the *<Engine ID>* of the authoritative agent involved. A trap message has two parts: a header and a trap Protocol Data Unit (PDU). The header contains the *<Security Engine ID>* and a *<username>* set in the trap configuration. When an agent sends a trap, these parameters in the trap header are checked against the details stored in the USM table. The trap is further processed only when there is a match.
- *Context Engine ID* – *<Context Engine ID>* is part of a trap PDU. It uniquely identifies a device which has sent the original trap message. *<Context Engine ID>* and *<Security Engine ID>* are identical in most cases.

- *USM Table* – SNMP managers receiving the traps needs to maintain the USM table (User-based Security Model) which has *<Security Engine ID>* and *<username>* as the key to verify the source of the trap messages.
- *Virtual IP Address for SNMP Proxy* – Paragon Insights is deployed as a Kubernetes application and it uses load balancers that exposes virtual IP addresses to external entities. When messages are routed through the load balancer, the source IP address of the trap message will be set to the IP address of the load balancer.

The load balancer has an option to retain the source IP address if you allot an exclusive virtual IP address for services that require the source IP address of SNMP agents to be preserved. Since in SNMP notifications, the source IP address is required for parsing the message, an exclusive virtual IP must be allotted for SNMP Proxy. The same virtual IP needs to be configured on the device or device groups as the target address.

Events such as interface flap in your network can create multiple inform or trap notifications. To determine the order of the notifications, Paragon Insights must capture the timestamp of devices sending SNMP notifications in nanosecond. However, SNMP protocol records the device timestamp (in seconds) as the notifications reach Paragon Insights. As SNMP notifications are sent through UDP, there is also no guarantee of the order of SNMP notifications.

To enable you to capture the order of SNMP notifications, the following fields are parsed from the SNMP Header of the notification packets.

- *Message ID*: Used by devices to identify the SNMP message and match responses to the message. Message ID can be found in the SNMP Header.

When you create a rule for SNMP trap or inform notifications, use the path `__msg_id__` in a field to collect message ID.

- *Request ID*: Used to identify the SNMP protocol data unit (PDU). Request ID can be found in the trap or inform PDU.

When you create a rule for SNMP trap or inform notifications, use the path `__req_id__` in a field to collect request ID.

- *SNMP Version*: Identifies the version number of the SNMP notification (2 or 3).

When you create a rule for SNMP trap or inform notifications, use the path `__version__` in a field to collect version number.

- *SNMP PDU Type*: Identifies the type of SNMP notification (v2c trap or inform request).

When you create a rule for SNMP trap or inform notifications, use the path `__pdu_type__` in a field to collect the PDU type.

The SNMP PDU type field data captured in the time-series database shows v2c for all SNMP trap notifications. You can check the version number to determine if the trap is generated using SNMP v2c or SNMP v3.

Configurations

IN THIS SECTION

- [Find the Engine Id | 62](#)
- [Trap Configurations | 63](#)
- [SNMPv3 Inform Configurations | 68](#)
- [Port Configuration | 70](#)
- [Rule Configuration | 71](#)

The following sections detail how to:

- ["Find the Engine Id" on page 62](#)
- ["Trap Configurations" on page 63](#)
- ["SNMPv3 Inform Configurations" on page 68](#)
- ["Rule Configuration" on page 71](#)
- ["Port Configuration" on page 70](#)

Find the Engine Id

Depending on if you configure devices to send trap or inform notifications, you need to first find the *<Engine ID>* of either the SNMP agent. You can refer to the sample commands below to find the engine id in Junos devices.

NOTE: The CLI command to find *<Engine ID>* varies from vendor-to-vendor.

To find the *<Engine ID>* of SNMP agents (devices) that are Junos-based platforms, enter the following command in CLI.

```
show snmp v3 engine-id
```

You will receive a HEX output as the device *<Engine ID>*.

Trap Configurations

You can configure a device to send trap notifications using SNMPv2c and SNMPv3.

The source IP address needs to be unique across all the devices as it uniquely identifies the device. The source IP address can only be configured under device while community name can be configured under both device and device group.

NOTE: In Paragon Insights, the SNMPv2c and SNMPv3 ingest and trap configurations share the same workflow.

To configure SNMP trap notifications at the device level:

1. Click the **Configuration > Device** option in the left navigation bar.
2. Click the add device button (+).
3. Enter the necessary values in the text boxes and select the appropriate options for the device.

The following table describes the attributes in the **Add a Device** window:

Table 5: Add Device(s) Page Details

Attributes	Description
Name	Name of the device. Default is hostname. (Required)
Hostname / IP Address / Range	Hostname or IP address of a single device. If you are providing a range of IP addresses, enter the IP address for the device that marks the start and end of the address range. (Required)
SNMP	
SNMP Port Number	Port number required for SNMP. The port number is set to the standard value of 161 .
SNMP Version	Select either v2c or v3 in the drop-down menu.

Table 5: Add Device(s) Page Details (Continued)

Attributes	Description
SNMP Community	<p>This field appears if you selected v2c in SNMP Version field.</p> <p>Enter an SNMP Community string if you configure SNMPv2c for trap notifications.</p> <p>In SNMPv2c, Community string is used to verify the authenticity of the trap message issued by the SNMP agent (devices such as routers, switches, servers, and so on).</p>
SNMPv3 Username	<p>This field appears if you selected v3 in SNMP Version field.</p> <p>Enter a username for trap notifications.</p>
Authentication None	<p>This field appears if you selected v3 in SNMP Version field.</p> <p>Enable this option on if you want to set SNMPv3 authentication to <i>None</i>.</p>
Protocol None	<p>This field appears if you selected v3 in SNMP Version field.</p> <p>Enable this option on if you want to set SNMPv3 protocol to <i>None</i>.</p>
SNMPv3 Authentication Protocol	<p>This field appears if you selected v3 in SNMP Version field and disabled <i>Authentication None</i>.</p> <p>Select an authentication protocol from the drop-down menu.</p> <p>SNMP authentication protocol hashes the SNMP <i>username</i> with the passphrase you enter. The hashed output is sent along with the trap notification message. Paragon Insights again hashes the <i>username</i> with the passphrase you entered for authentication. If the output matches, the trap notification is further processed.</p>

Table 5: Add Device(s) Page Details (Continued)

Attributes	Description
SNMPv3 Authentication Passphrase	<p>This field appears if you selected v3 in SNMP Version field and disabled <i>Privacy None</i>.</p> <p>Enter a passphrase for SNMPv3 authentication.</p>
SNMPv3 Privacy Protocol	<p>Select a privacy protocol from the drop-down menu.</p> <p>Privacy algorithm encrypts the trap notification message with the protocol passphrase so that the message cannot be read by an unauthorized application in the network.</p>
SNMPv3 Privacy Passphrase	<p>This field appears if you selected v3 in SNMP Version field and disabled <i>Privacy None</i>.</p> <p>Enter a passphrase to encrypt the trap notification.</p>
Context Engine ID	<p>This field appears if you selected v3 in SNMP Version field.</p> <p>The <i>Engine ID</i> must be set to engine-id of the SNMP agent.</p>
Source IP Address	<p>Enter the source IP address of the device.</p> <p>This field is mandatory for SNMPv2c traps and optional for SNMPv3 traps and informs.</p> <p>If you use NAT or an SNMP Proxy, the virtual IP address you configure for the SNMP Proxy must be set as the source IP address.</p> <p>To set virtual IP address for SNMP Proxy, go to Settings > Deployment in the left navigation bar. In the Loadbalancer page, enter the virtual IP adres and click Save and Deploy button.</p>

4. Click **Save** to commit the configuration or click **Save and Deploy** to deploy the configuration in Paragon Insights.

To configure SNMP trap notifications at the device-group level:

1. Click the **Configuration > Device Group** option in the left navigation bar.
2. Click the add group button (+).
3. Enter the necessary values in the text boxes and select the appropriate options for the device group.

The following table describes the attributes in the **Add a Device Group** window:

Table 6: Add Device Group Page Details

Attributes	Description
Name	Name of the device group. (Required)
Description	Description for the device group.
Devices	<p>Add devices to the device group from the drop-down list. (Required)</p> <p>Starting in Paragon Insights Release 4.0.0, you can add more than 50 devices per device group. However, the actual scale of the number of devices you can add depends on the available system resources.</p> <p>For example, consider that you want to create a device group of 120 devices. In releases earlier than release 4.0.0, it is recommended that you create three device groups of 50, 50, and 20 devices respectively. With Paragon Insights Release 4.0.0, you just create one device group.</p>
SNMP	
SNMP Port Number	Port number required for SNMP. The port number is set to the standard value of 161 .
SNMP Version	<p>Select either v2c or v3 in the drop-down menu.</p> <ul style="list-style-type: none"> • If you select v2c, the SNMP Community name field appears. The string used in v2c authentication is set to <i>public</i> by default. It is recommended that users change the community string. • If you select v3, you are given an option to set username, authentication and privacy methods, and authentication and privacy secrets.

Table 6: Add Device Group Page Details (Continued)

Attributes	Description
Notification Ports	<p>Enter notification ports separated by comma.</p> <p>Paragon Insights listens on these notification ports for traps and inform notification messages from device groups.</p>
SNMP Community	<p>This field appears if you selected v2c in SNMP Version field.</p> <p>Enter an SNMP Community string if you configure SNMPv2c for traps and ingest.</p> <p>In SNMPv2c, Community string is used to verify the authenticity of the trap notification messages issued by the SNMP agent.</p>
SNMPv3 Username	<p>This field appears if you selected v3 in SNMP Version field.</p> <p>Enter a username for trap notifications.</p> <p>The USM configuration configured under device-groups is shared among devices configured under the same device-group.</p>
Authentication None	<p>This field appears if you selected v3 in SNMP Version field.</p> <p>Enable this option on if you want to set SNMPv3 authentication to <i>None</i>.</p>
Privacy None	<p>This field appears if you selected v3 in SNMP Version field.</p> <p>Enable this option on if you want to set SNMPv3 privacy protocol to <i>None</i>.</p>
SNMPv3 Authentication Protocol	<p>This field appears if you selected v3 in SNMP Version field and disabled <i>Authentication None</i>.</p> <p>Select an authentication protocol from the drop-down menu.</p> <p>SNMP authentication protocol hashes the SNMP <i>username</i> with the passphrase you enter. The hashed output is sent along with the trap notification message. Paragon Insights again hashes the <i>username</i> with the passphrase you entered for authentication. If the output matches, the trap notification is further processed.</p>
SNMPv3 Authentication Passphrase	<p>This field appears if you selected v3 in SNMP Version field and disabled <i>Privacy None</i>.</p> <p>Enter a passphrase for SNMPv3 authentication.</p>

Table 6: Add Device Group Page Details (Continued)

Attributes	Description
SNMPv3 Privacy Protocol	<p>Select a privacy protocol from the drop-down menu.</p> <p>Privacy algorithm encrypts the trap notification message with the protocol passphrase so that the message cannot be read by an unauthorized application in the network.</p>
SNMPv3 Privacy Passphrase	<p>This field appears if you selected v3 in SNMP Version field and disabled Privacy None.</p> <p>Enter a passphrase to encrypt the trap notification.</p>
Logging Configuration	
SNMP Notification	<p>Paragon Insights Release 4.0.0 supports collecting log data for SNMP notification. You can collect different severity levels of logs for snmp-notification service in a device group.</p> <p>Use these fields to configure which log levels to collect:</p> <p>Global Log Level From the drop-down list, select the level of the log messages that you want to collect for every running Paragon Insights service for the device group. The level is set to error by default.</p> <p>Log Level for specific services Select the log level from the drop-down list for any specific service that you want to configure differently from the Global Log Level setting. The log level that you select for a specific service takes precedence over the Global Log Level setting.</p>

4. Click **Save** to commit the configuration or click **Save and Deploy** to deploy the configuration in Paragon Insights. For information on how to use the device group cards, see *Monitor Device and Network Health*.

SNMPv3 Inform Configurations

To enable devices to send inform notifications, you must configure SNMPv3 USM user(s).

To create USM users in Paragon Insights:

1. Navigate to **Settings > Ingest**.
2. Select the **SNMP Notification** tab on the Ingest Settings page.

3. Click the **Usm Users** section.
4. Click the plus (+) icon to add a USM user.
5. In the Add USM User page, enter the username, select the authentication and the privacy protocols, and enter passphrases.

If you enabled Authentication None and Privacy none, the protocol menu and passphrase fields do not appear.

6. Click **Save** to only save the configuration and **Save and Deploy** to deploy the configuration in Insights.

After adding USM users, you can configure the following details in the Add Device(s) page in Device Configuration or Add Device Group page in Device Group Configuration.

Table 7: SNMP Configuration for Informs

Attributes	Description
SNMP	
SNMP Port Number	Port number required for SNMP. The port number is set to the standard value of 161 .
SNMP Version	Select v3 in the drop-down menu.
Notification Ports (Device Groups only)	Enter notification ports separated by comma. Paragon Insights listens on these notification ports for traps and inform notification messages from device groups.
Context Engine ID (Devices only)	This field appears if you selected v3 in SNMP Version field. The <i>Engine ID</i> must be set to engine-id of the SNMP agent.

Table 7: SNMP Configuration for Informs (Continued)

Attributes	Description
Source IP Address (Devices only)	<p>This field appears if you selected v3 in SNMP Version field.</p> <p>Enter the source-IP-address of the device.</p> <p>If you use NAT or an SNMP Proxy, the virtual IP address you configure for the SNMP Proxy must be set as the source IP address.</p> <p>To set virtual IP address for SNMP Proxy, go to Settings > Deployment in the left navigation bar. In the Loadbalancer page, enter the virtual IP address and click Save and Deploy.</p>

Starting with Paragon Insights 4.2.0, you can collect `_device_timestamp_` as a raw data metric. Device timestamp (in seconds) denotes the time when the inform notification is sent from a device.

NOTE: Device timestamp for an inform notification is captured by the SNMP sensor only if you configured authentication for an inform message.

The device timestamp data for SNMP trap notifications are the same as the *received time* metric in Paragon Insights. The device timestamp for traps and the *time* metric for trap notifications denote the time when the SNMP notification is received by Paragon insights.

Port Configuration

By default, Paragon Insights listens for traps and informs in the standard SNMP trap port 162. This can be changed if needed either at the global level which is applicable to all device groups or at the device group level applicable to a specific device group.

Port configured under ingest will apply to all device groups. Trap and Inform messages received through any other port are discarded.

To configure port number at the ingest level:

1. Navigate to **Settings > Ingest** in the left-nav bar.
2. Select the **SNMP Notification** tab on the Ingest Settings page.
3. In the **Port** section, enter the port number.
4. Click **Save** to only save the configuration and **Save and Deploy** to deploy the configuration in Insights.

Port configured under device group will apply to only a specific device group. Traps and informs received through any other port are discarded. To configure port numbers at the device group level, see [Table 6 on page 66](#).

Rule Configuration

Once the device is configured to send traps or inform notification, you must configure a rule on the device with SNMP trap so that, Paragon Insights can process traps from the device. In device groups, you can apply a playbook instance that has the snmp-notification rule. When you configure SNMP notification in any rule, you must select the MIB name you want to monitor. Go to [Juniper MIBS Explorer](#) to browse MIB files for Junos devices and [Cisco MIBS Locator](#) to browse MIB files for Cisco devices.

The following example shows how you can configure a rule with SNMP notification to send alerts if an interface comes up for the *chassis.interfaces/* topic.

NOTE: It is assumed that you have configured the device or device group for SNMP trap notification. See ["Paragon Insights Pull-Model Ingest Methods" on page 109](#) to configure SNMP trap notifications in devices or device groups.

To configure a rule under topic *system.trap/*:

1. Go to **Configuration > Rules**.
2. Click the **Add Rules** button in the Rules page.

Enter the rule name in the *topic/rule-name* format in the Rule field and description in the Description field. For example, *chassis.interfaces/linkup*.
3. Click **Add Sensor** button in Sensors tab.
4. Enter a name in the Sensor Name field and select **SNMP Notification** from the drop-down menu in Sensor Type.
5. Enter notification name in *MIB-Name::Notification Name* format.

For example, **IFMIB::linkDown**.
6. Click **Add Field** button in the Fields tab.

The fields for SNMP Notification rule can be derived as described:

- Variables (varbinds) for the given trap.

The variables of the trap can be defined as fields. The following steps use the example *IfAdminStatus* as varbind and *IF-MIB:linkDown* as the snmp-notification.

a. Enter *IfAdminStatus* in the Field Name.

b. Select *Integer* as **Field Type**.

The **Field Type** you enter in the GUI must be same as the type defined in the MIB File.

c. Select *Sensor* as **Ingest Type** (field source).

The Ingest Type (field source) must be set to sensor.

d. Select the sensor name from the drop-down menu under **Sensor**.

The sensor name is the name you entered for the snmp-notification sensor.

e. Enter *IfAdminStatus* as sensor path.

The **Path** must be set the to the variable (varbind) name defined in the MIB file.

To add a second field for *IfOperStatus* as variable (varbind) for a given snmp-notification, follow the steps described here but change the field name and sensor path to *IfOperStatus*.

7. Click **Save** to commit the rule or **Save & Deploy** to deploy the rule in Paragon Insights.

You can see the new topic name and rule in the list of existing rules.

You can also configure triggers or functions based on the fields you add. See how to create a new rule in GUI as explained in *Paragon Insights Rules and Playbooks*.

You must include this rule in a playbook and apply the playbook instance on a device or device group.

To check the new SNMP notifications sent by device groups, log into Paragon Insights server as a root user and type the following command.

```
healthbot cli --device-group healthbot -s influxdb
```

You can track new entries sent by SNMP trap notifications to the Paragon Insights server for the fields (for example, *IfAdminStatus*) you configured.

Release History Table

Release	Description
4.0.0	Starting in Paragon Insights Release 4.0.0, you can clone an existing NetFlow template.
4.0.0	Starting with Paragon Insights Release 4.0.0, you can clone an existing Syslog pattern.
4.0.0	Starting with Paragon Insights Release 4.0.0, you can clone an existing Syslog pattern set.

4.0.0	Starting in Paragon Insights Release 4.0.0, you can configure the pattern for parsing the header portion of a syslog message.
4.0.0	Paragon Insights Release 4.0.0 supports inform and trap notifications that are sent by devices in the network for fault management.
4.0.0	Starting in Paragon Insights Release 4.0.0, you can add more than 50 devices per device group.
4.0.0	Paragon Insights Release 4.0.0 supports collecting log data for SNMP notification.
3.1.0	Starting with Paragon Insights Release 3.2.0, Paragon Insights supports sFlow (v5)
3.1.0	Starting with release 3.1.0, Paragon Insights supports the gRPC Network Management Interface (gNMI) for OpenConfig communication.
3.0.0	Starting with Release 3.0.0, Paragon Insights (formerly HealthBot) supports NetFlow natively as another ingest method
2.1.0	starting with Release 2.1.0, Paragon Insights also supports syslog natively as another data collection method
2.0.0	In Paragon Insights releases prior to 3.1.0, OpenConfig could only be used on Junos OS and certain Cisco devices.

Understand Inband Flow Analyzer 2.0

IN THIS SECTION

- [Device Configuration | 74](#)
- [Paragon Insights Configuration | 76](#)

Inband Network Telemetry (INT) is a vendor-neutral network monitoring framework that provides per-hop granular data in the forwarding (data) plane. INT allows you to observe changes in flow patterns caused by microbursts, packet transmission delay, latency per node, and new ports in flow paths.

Inband Flow Analyzer (IFA) 2.0 is an implementation of INT in Junos OS switches to collect flow data and export the data to external collectors for per-hop or end-to-end analysis. IFA uses probe packets to collect data such as per-hop latency, per-hop ingress and egress ports, packet Receive (RX) timestamp (in seconds), queue ID, congestion, and egress port speed. The IFA packets traverse the same path in the network and use the same queues as the packets in the forwarding plane. So, the IFA packets experience similar latency and congestion as the packets in the forwarding plane.

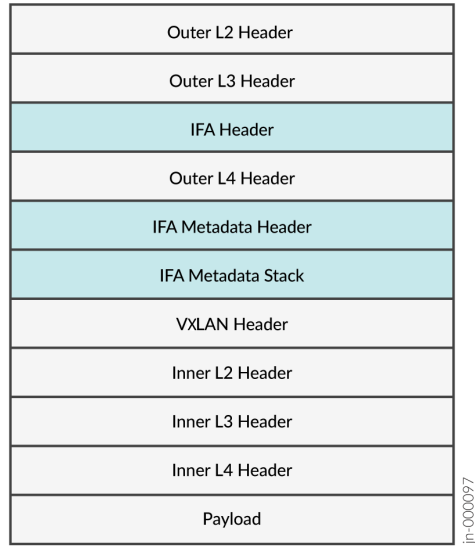
Device Configuration

The QFX5120-32C and QFX5120-48Y devices support Inband Network Telemetry (INT) using IFA 2.0. The IFA probe packets collect flow metrics and export the data in the Internet Protocol Flow Information Export (IPFIX) format. Starting with Release 4.2.0, Paragon Insights supports analysis of the IPv4 Virtual Extensible LAN (VXLAN) flow data using the IFA sensor. Paragon Insights identifies VXLAN flows if the standard VXLAN port 4789 is present as the destination port in the Outer L4 Header (Layer 4 Header). The format of the IFA 2.0 packet with the VXLAN flow data is shown in [Figure 8 on page 75](#).

NOTE: IFA uses revenue ports to export data to collectors. You cannot use management ports to export IFA data.

See [IFA Configurations and Design Considerations](#) for detailed information on supported platforms and the device configuration required to use IFA.

Figure 8: Format of VXLAN IFA 2.0 Packet



IFA probe packets use three nodes that have separate functionality as they collect flow information:

- *IFA Initiator Node (ingress node)*—Samples the IPv4 VXLAN traffic, converts packets to IFA format by adding an IFA header, and updates IFA probe packet with the Initiator Node metadata. The IFA Header has the total maximum length allowed for the IFA Metadata Stack. The metadata stack is where each node adds its respective hop-specific metadata.
- *IFA Transit Node*—Identifies IFA packets and appends metadata into the metadata stack of the packet. A transit node checks the current length against the total maximum length in the IFA Header. If the current length equals or exceeds the maximum length, the Transit Node does not append its metadata to the IFA Metadata Stack.
- *IFA Terminating Node (egress node)*—Appends its metadata and exports a copy of the flow data to the IFA 2.0 application (the IFA firmware). The IFA application adds the egress port number, converts the packets into IPFIX format, and sends them to a collector such as Paragon Insights.

See [IFA Configurations and Design Considerations](#) for more information.

NOTE: You must configure the IFA Initiator Node, IFA Transit Node, and IFA Terminating Node in the QFX5120-32C and QFX5120-48Y switches.

Paragon Insights Configuration

In Paragon Insights, you must perform the following tasks:

1. Configure IFA flow IP address in devices and configure IP address of the deploy node and the UDP port in the device group. Paragon Insights deploys the IFA ingest on the configured deploy node.

See *Manage Devices, Device Groups, and Network Groups* for more information.

2. Create a new rule for IFA ingest.

See *Paragon Insights Rules and Playbooks* for more information.

3. Create a playbook and deploy the playbook instance in device groups.

See *Paragon Insights Rules and Playbooks* for more information.

4. Configure device details such as device name and device ID in the ingest. See "[Configure Device Details for Inband Flow Analyzer Devices](#)" on page 79.

Paragon Insights supports hb_ifa_v2_0 as the IFA sensor name. The IFA sensor supports fields described in [Table 8 on page 77](#).

Table 8: IFA Sensor Fields

Field	Key Field	Data Type	Description
source_ip	Yes	String	IP address of the Initiator Node from which the IFA flow packets originate.
source_port	Yes	String	Source port of the Initiator Node from which the IFA packet originates.
dest_ip	Yes	String	IP address of the Terminating Node.
dest_port	Yes	String	Destination port of the Terminating Node that exports the IFA packets.
proto	Yes	String	Value of the protocol used for the IFA flow.
hop	Yes	String	<p>The hop field denotes the number of hops that the the IFA packet traversed. If there are n nodes, the hop value starts with 1 for the Initiator node, 2 for the Transit node, and so on until it reaches the Terminating node that is assigned a value of n.</p> <p>NOTE:</p> <p>The IFA sensor can additionally assign the hop value 65,535 to describe end-to-end latency and the complete IFA flow path.</p> <p>In Paragon Insights rules, the hop field captures the sequence number (hop value) at each hop.</p>
node_id	No	String	<p>Device ID of the IFA Initiator node, the IFA Transit node, or the IFA Terminator node, when the hop field's value is not 65,535. The device ID is present in the IFA Metadata Stack.</p> <p>When the hop field's value is 65,535, the node_id field denotes the complete path taken by the IFA probe packet.</p>

Table 8: IFA Sensor Fields (Continued)

Field	Key Field	Data Type	Description
node_name	No	String	Displays name of the IFA node associated with the <i>node_id</i> , if you previously configured Paragon Insights to display the <i>node_name</i> . If you didn't configure Paragon Insights to display the <i>node_name</i> , the <i>node_id</i> is displayed.
ingress_port	No	String	Ingress port of the node through which the IFA flow enters.
egress_port	No	String	Egress port of the node through which the IFA flow exits.
egress_portspeed	No	Unsigned integer 32	Speed (in Gigabits per second) of the egress port.
congestion_bits	No	Unsigned integer 32	Congestion bit that indicates if an IFA packet experienced congestion or not.
queue_id	No	Unsigned integer 32	Identifier (ID) of the queue taken by the IFA packets in a node.
residence_time_ns	No	Unsigned integer 32	Time taken (in nanoseconds) by the IFA packet within a node.
rx_ts_ns	No	Unsigned integer 64	Receive timestamp value when the IFA probe packet enters a node.
latency	No	Unsigned integer 64	Difference between the received timestamp of the current node and the previous node, when the hop field's value is not 65,535. When the hop field's value is 65,535, the latency field denotes the end-to-end latency of the complete path.

Paragon Insights ingests the IFA data as IPFIX records and creates multi-row entries in the time-series database (TSDB) for each IPFIX record. The TSDB rows capture per hop details such as:

- Ingress and egress ports

- Latency
- Receive packet (RX) timestamp
- Sequence number that increments at each hop
- A record of the end-to-end latency from the Initiator node to the Terminating node

RELATED DOCUMENTATION

| [Paragon Insights Pull-Model Ingest Methods](#) | 109

Configure Device Details for Inband Flow Analyzer Devices

You can access the IFA Devices page from **Settings>Ingest**. In the Ingest Settings page, click the IFA Devices menu. In Paragon Insights Release 4.3, you can assign device specific details, such as device name and device ID, for the Inband Flow Analyzer (IFA) devices. Paragon Insights maps the device names to their respective device ID. The device ID you enter corresponds to the device ID in the Junos device configuration you earlier completed to enable IFA. Monitoring the end-to-end path of IFA devices in a flow is more human readable. You can view the device name you configure in the IFA Devices page instead of the device ID. Paragon Insights displays this name as `node_name` in the time series database field table.

To assign a name to an IFA device:

1. Click the add icon (+).

The Create IFA Device page appears.

2. Enter the device ID.

Ensure that the device ID matches the device ID you entered for IFA devices. See [IFA Configurations and Design Considerations](#) for more information.

3. Do one of the following:

- Click **Save** to only save the configuration.

Paragon Insights saves the configuration to assign a name to the device but you cannot view the device name while monitoring the complete path of the IFA probe packets.

- Click **Save and Deploy** saves and deploys the configuration. You can see the device name you configured as `node_name` in the complete path of the IFA probe packets.

View the complete path of IFA probe packets in a flow using Grafana. See *Monitor Network Device Health Using Grafana* for more information.

RELATED DOCUMENTATION

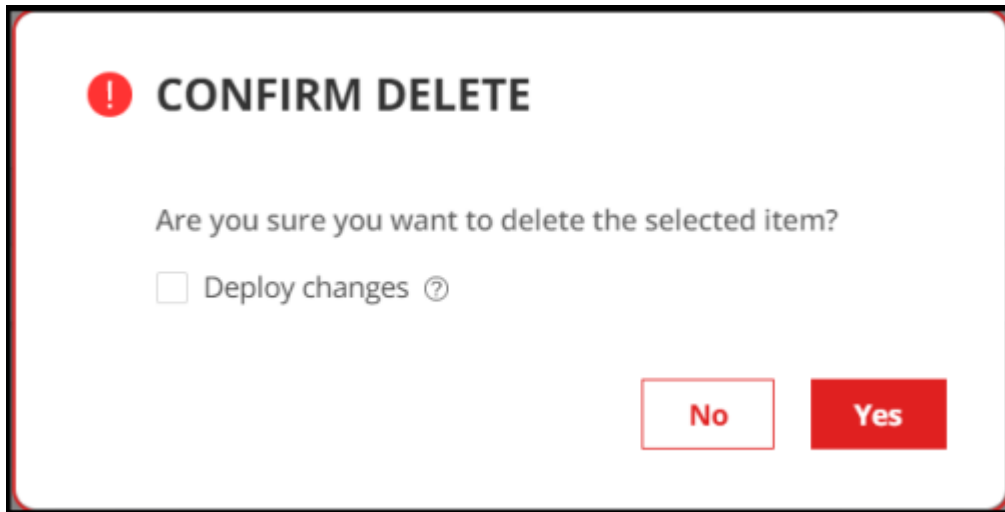
[Understand Inband Flow Analyzer 2.0 | 73](#)

Delete an Inband Flow Analyzer Device

To delete an IFA device:

1. Click **Settings** > **Ingest** from the left-nav bar.
The **Ingest Settings** page is displayed.
2. Click the **IFA Device** tab to view the **IFA Devices** page.
3. Select the device that you want to delete, and click the **delete (trash can)** icon.
The **CONFIRM DELETE** pop-up appears.

Figure 9: Confirm Delete Pop-up



4. Do any one of the following:
 - Click **Yes** to delete the IFA device from the database. However, the changes are not applied to the ingest service.

NOTE:

- We recommended that you do not delete an IFA device that is currently in use.
 - After you delete an IFA device from the database, you cannot associate that IFA device with another device group even if you have not deployed changes.
 - You can also deploy changes to the ingest service or roll back the changes that you have already deleted, from the **PENDING CONFIGURATION** page. For more information, see *Commit or Roll Back Configuration Changes in Paragon Insights*.
- Select the **Deploy changes** check box and then click **Yes** to delete the IFA device from the database, and to apply the changes to the ingest service.
 - (Optional) Click **No** to cancel this operation.

Understand Bring Your Own Ingest

IN THIS SECTION

- [Benefits](#) | 82

Paragon Insights provides Bring Your Own Ingest (BYOI) default plug-ins and support for BYOI custom plug-ins. BYOI plug-ins ingest telemetry data that is stored in third-party sources such as a data lake or external databases. You can export such telemetry data that you collected through the BYOI plug-ins and store it in the Paragon Insights time series database (TSDB).

Bring Your Own Ingest Plug-ins include an input plug-in that is developed by the user and the output plug-in developed by Juniper Networks. The BYOI input plug-in streams data from different data sources (Kafka) that use different data models (OpenConfig or NETCONF YANG), data encoding (based on Extended Markup Language [XML] or JavaScript Object Notation [JSON]), data security, and messaging buses (Kafka), and sends the data to the output plug-in. The output plug-in converts that data into the line protocol format and writes the data to the Paragon Insights TSDB.

Paragon Insights supports two types of BYOI plug-ins:

- *Default plug-ins*—Use default plug-ins to measure metrics that are unique to your network. You can work with Juniper Networks to develop default BYOI plug-ins. To load a default plug-in, you require Paragon Insights Release 4.2 or later.

Juniper Network develops and sends you the default plug-ins, with the Kubernetes YAML files and the plug-in configurations that are included as a compressed tar file. For more information on the workflow to deploy a default plug-in, see ["Bring Your Own Ingest Default Plug-in Workflow" on page 83](#).

- *Custom plug-ins*—You can use custom plug-in when you want to stream pre-existing telemetry data to Paragon Insights for analysis. You must develop the BYOI plug-in, build the BYOI plug-in ingest image, and load the plug-in image and Kubernetes YAML file for the plug-in to the Paragon Insights server.

To load a custom plug-in, you require Paragon Insights Release 4.2 or later. After you successfully load the custom plug-in, it is listed in the Custom Plugins tab of the Bring Your Own Ingest Plugins page.

For more information on the workflow to deploy a custom plug-ins, see ["Bring Your Own Ingest Custom Plug-in Workflow" on page 89](#).

Benefits

Deploying bring your own ingest plug-ins has the following benefits:

- Reduces the cost of collecting telemetry data from devices by exporting previously collected data to Paragon Insights.
- Enables you to use all Paragon Insights features—such as custom or default rules, playbooks, reports, graphs, network health view, and more—on the external data you ingest into Paragon Insights.

RELATED DOCUMENTATION

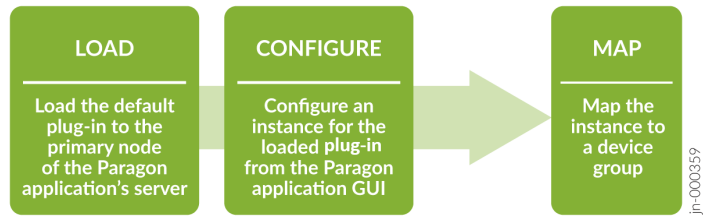
[Load Bring Your Own Ingest Default Plug-ins | 85](#)

[Build and Load BYOI Custom Plug-in Images | 91](#)

Bring Your Own Ingest Default Plug-in Workflow

The workflow to load and deploy the BYOI default plug-in involves the steps that are illustrated in [Figure 10 on page 84](#).

Figure 10: Default BYOI Plug-in Workflow



The following tasks comprise the end-to-end workflow to use the BYOI default plug-in:

1. Load the default plug-in to the primary node of the Paragon Insights server. See "[Load Bring Your Own Ingest Default Plug-ins](#)" on page 85.
2. Create an instance of the plug-in you earlier loaded. See "[Configure Bring Your Own Ingest Default Plug-in Instances](#)" on page 86.
3. Map the plug-in instance to sensors and device groups that can then use the plug-in. See "[Configure Ingest Mapping for Default BYOI Plug-ins](#)" on page 88.
4. Add the default plug-in as a sensor in a rule. See *Create a New Rule Using Paragon Insights GUI in Paragon Insights Rules and Playbooks*.
5. Add the rule to a playbook. See *Create a New Playbook Using Paragon Insights GUI in Paragon Insights Rules and Playbooks*.
6. Deploy a playbook instance. See *Manage Playbook Instances in Paragon Insights Rules and Playbooks*.

RELATED DOCUMENTATION

[Understand Bring Your Own Ingest](#) | 81

Load Bring Your Own Ingest Default Plug-ins

To load a BYOI default plug-in, we recommend that you have a multinode installation of Paragon Insights in your proof of concept or production systems. A root or a non-root user can enter the commands to load your BYOI default plug-in on your server that hosts the Paragon Insights application.

1. Log in to the primary node of the Paragon Insights server using your server credentials.
2. Enter the following command to list the loaded default plug-ins.

```
user@primary-node# sudo healthbot list-plugins -d
```

3. Enter the following command to load a BYOI plug-in.

```
user@primary-node# sudo healthbot -v load-plugin -n plugin-name
```

For example, `user@primary-node# sudo healthbot -v load-plugin -n tlive_kafka_oc`.

An authentication prompt appears.

4. Enter your GUI login credentials for Paragon Insights.
A CLI message confirms that the plug-in is successfully loaded.

You can see the loaded default plug-in in the Paragon Insights GUI. The default plug-in appears as a new tab in the Default Plugins page (**Settings > Ingest > BYO Ingest Plugins**).

5. Login to the Paragon Insights GUI and click **Settings>BYO Ingest Plugins** to verify that the plug-in is visible in the GUI.

You can see the loaded default plug-in listed in the Default Plugins page.

After the plug-in is loaded successfully, you must create an instance of the default plug-in and configure ingest mapping.

What's Next

- ["Configure Bring Your Own Ingest Default Plug-in Instances" on page 86](#)
- ["Configure Ingest Mapping for Default BYOI Plug-ins" on page 88](#)

Configure Bring Your Own Ingest Default Plug-in Instances

You must configure a new instance of the Bring Your Own default plug-in Ingest (BYOI) that you previously loaded.

To configure a default plug-in instance:

1. Select **Configuration > Ingest**.

The Ingest Settings page appears.

2. Click the **BYO Ingest Plugins** tab.

If you loaded the default plug-in, you can see its name as a tab on the Default Plugins page.

3. Click **+ New Instance**.

The Add a *default-plugin-name* Instance page appears. For example, Add a *tlive-kafka-oc* Instance page.

4. Enter details as described in [Table 9 on page 87](#).

NOTE:

Fields marked with an asterisk (*) are mandatory.

5. Do one of the following:

- **Save**—Save the default plug-in instance configuration but do not deploy the configuration.

You can use this option to save make multiple changes in Paragon Insights configurations and commit the changes in bulk or to roll back the changes. See *Commit or Roll Back Configuration Changes in Paragon Insights* for more information.

- **Save & Deploy**—Save and deploy the configuration.

Paragon Insights creates an instance for the BYOI plug-in and deploys the configuration.

Table 9: Fields on the Add a Default-Plugin-Name Instance Page

Field	Description
Name	<p>Enter a name for the default ingest plug-in instance.</p> <p>You can enter a name that follows the regular expression <code>^[a-zA-Z][a-zA-Z0-9_-]*\$</code> with maximum 64 characters.</p> <p>For example, t1 as name for an instance of the Tlive-kafka-oc plug-in.</p>
Brokers	<p>Enter one or more broker names separated by commas.</p> <p>If you do not enter a broker, the default Kafka broker is healthbot.</p>
Topics	<p>Enter healthbot as the topic to which the plugin subscribes.</p>
Authentication	
SASL Username	<p>Enter the username for authentication.</p>
Password	<p>Enter a password that has 6 to 128 characters long.</p> <p>The password must contain a combination of uppercase characters, lowercase characters, numbers, and special characters. Paragon Insights supports all keyboard special characters such as comma, asterisk, ampersand and so on.</p>

Table 9: Fields on the Add a Default-Plugin-Name Instance Page (*Continued*)

Field	Description
TLS	
CA Profile Name	Enter the name of the trusted certificate authority (CA).
Local Certification Profile Name	Enter the name of the self-signed certificate created for Paragon Insights.
Skip Certification Chain and Host Verification	<p>Toggle the switch on if you want to skip verifying the integrity of the CA certificate.</p> <p>NOTE: The connection between the ingest plug-in and the external data source is encrypted even if you skip verification of the certificate.</p>

After you complete the instance configuration, you must configure the ingest mapping for the default plug-in instance. See "[Configure Ingest Mapping for Default BYOI Plug-ins](#)" on page 88 for more information.

RELATED DOCUMENTATION

[Understand Bring Your Own Ingest](#) | 81

Configure Ingest Mapping for Default BYOI Plug-ins

For default ingest plug-ins, you must configure the sensors and device groups that can use the plug-in.

To configure the ingest mappings for a default plug-in:

1. Go to **Configuration > Ingest > BYOI Plugins**.
2. Click **+New Mapping** in the Ingest Mapping tab.

The Add a New Ingest Mapping page appears.

NOTE: You can configure multiple ingest mappings for the same default plug-in.

3. Enter the details as described in [Table 10 on page 89](#).

Table 10: Attributes in Add a New Ingest Mapping Page

Attributes	Description
Name	Enter a name for the ingest mapping. The name is an instance identifier of the ingest mapping.
Sensor Type	Select the type of sensor to be used for the plug-in from the list.
Plugin Name	Enter the name of the default ingest plug-in.
Constraint to Device Groups	Select device groups from the list. The ingest and sensor mapping is applied to only the selected device groups.

4. Do one of the following:
- **Save** – Save your ingest mapping but do not deploy the updated configuration. You can use this option when, for example, you are making several changes and want to deploy all your updates at the same time later.
 - **Save & Deploy** – Save the ingest mapping configuration and deploy the configuration in your production environment.

You can use BYOI ingest in rules after you configure the ingest mapping for a default plug-in.

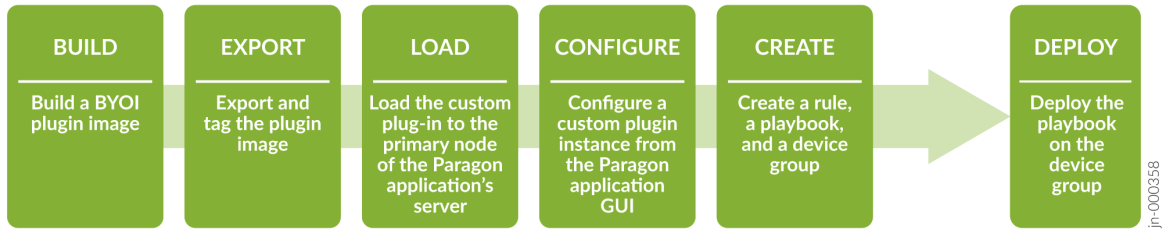
RELATED DOCUMENTATION

| [Understand Bring Your Own Ingest | 81](#)

Bring Your Own Ingest Custom Plug-in Workflow

The workflow to build, load, and to deploy the BYOI custom plug-in involves the steps that are illustrated in [Figure 11 on page 90](#).

Figure 11: Custom Plug-in Deployment Workflow



You must perform the following tasks to use custom BYOI plug-in in Paragon Insights:

1. Custom BYOI plug-in requires an ingest image file and a Kubernetes YAML file. Build the ingest image, configure the Kubernetes YAML file, and load the image and YAML files. See "[Build and Load BYOI Custom Plug-in Images](#)" on page 91.
2. Create an instance of the plug-in you earlier loaded. See "[Configure Bring Your Own Ingest Custom Plug-in Instances](#)" on page 102.
3. Create rules, playbooks, and device groups that use the custom plug-in. See "[Use the Sample Rule and Playbook Configurations for BYOI Custom Plug-ins](#)" on page 105.

RELATED DOCUMENTATION

[Understand Bring Your Own Ingest](#) | 81

Build and Load BYOI Custom Plug-in Images

IN THIS SECTION

- [Use JSON Configuration File Attributes in Ingest Image](#) | 93
- [Create a Shell Script for Configuration Updates](#) | 96
- [Tag and Export the BYOI Custom Plugin Image](#) | 96
- [Configure Kubernetes YAML File](#) | 96
- [\(Optional\) Assign Virtual IP Address to Plugin](#) | 100
- [Load the BYOI Custom Plugin](#) | 102

To send data to Paragon Insights using Bring Your Own Ingest (BYOI) custom plug-ins, you must code the input plug-in and create a plug-in ingest image file with a shell script and a process file. The ingest image also contains a Kubernetes YAML file for the ingest container. Paragon Insights executes the shell script when configurations of the BYOI ingest and device group (mapped to the ingest) change. The Kubernetes YAML file contains configurations that instruct the Kubernetes engine to start and to stop the service for a BYOI plugin ingest.

The workflow to build and load the BYOI custom plug-in image is as follows:

1. Create an ingest image to write data to the Paragon Automation database and a shell script for configuration updates.
 - The process file (a Python file in our example) parses attributes in the JSON configuration file to send data as fields to the Paragon Insights database.

See ["Use JSON Configuration File Attributes in Ingest Image" on page 93](#) for an example Python script.
 - Paragon Insights informs the BYOI plug-in to execute the shell script when you change configurations.

See ["Create a Shell Script for Configuration Updates" on page 96](#) for an example shell script.
2. Tag the image file and export the image as a compressed tar file.

See ["Tag and Export the BYOI Custom Plugin Image" on page 96](#) for commands to tag and export the image file.
3. Modify the Kubernetes Jinja template to create a YAML file for the Kubernetes container pod. The container pod is where the BYOI ingest is deployed in Paragon Insights.

See ["Configure Kubernetes YAML File" on page 96](#) for a sample Kubernetes Jinja template file.
4. Assign a different IP address if you want the BYOI plugin to be accessible for external applications.

See ["\(Optional\) Assign Virtual IP Address to Plugin" on page 100](#) for a Kubernetes Jinja template in which you can assign a custom virtual IP address for the BYOI plug-in.
5. Load the compressed tar file and Kubernetes YAML file to the Paragon Insights primary node.

See ["Load the BYOI Custom Plugin" on page 102](#) to load the BYOI plug-in image file and the Kubernetes YAML file.

WHAT'S NEXT

["Configure Bring Your Own Ingest Custom Plug-in Instances" | 102](#)

["Use the Sample Rule and Playbook Configurations for BYOI Custom Plug-ins" | 105](#)

Use JSON Configuration File Attributes in Ingest Image

IN THIS SECTION

- [Decode Device Password | 95](#)

You can create a Python file, such as the following example file, and include it in the BYOI plug-in image. When you run the image in a Kubernetes container, Paragon Insights executes the Python file. The following sample Python file uses the attributes described in [Table 11 on page 94](#) to send a key (a random integer between 0 and 9 in the example file) for the measurement (topic/rule/sensor_name/byoi) to the database.

```
import requests
import time
import random
import os
import json

# read tand_host, tand_port from env vars
tand_host = os.environ.get('TAND_HOST', 'localhost')
tand_port = os.environ.get('TAND_PORT', '3000')

# read device, plugin, rule related attributes from config json
with open('/etc/byoi/config.json', 'r') as f:
    config_json = json.load(f)
# input, device, sensor as lists. modify index as needed. Using 0 for all idxes
database = config_json['hbin']['inputs'][0]['plugin']['config']['device'][0]['healthbot-storage']
['database']
measurement = config_json['hbin']['inputs'][0]['plugin']['config']['device'][0]['sensor'][0]
['measurement']

# Construct post request and data
url = 'http://{}:{}/write?db={}'. \
    format(tand_host, tand_port, database)
data = '{} {} {}'
metric = 'key'
```

```

while True:
    fields = '{}={}'.format(metric, random.randint(0,9))
    timestamp = int(time.time()) * 1000000000
    x = requests.post(url, data=data.format(measurement, fields, timestamp))
    time.sleep(10)

```

In the Python file, use the following URL format to send data to Paragon Insights.

```

url = 'http://{}/write?db={}'. \
format(tand_host, tand_port, database)

```

The entry in the database (db) field must follow the syntax *database-name:device-group-name:device-name*.

The line protocol (post body) contains a string in the following format.

```

data = '{} {} {}'.format(measurement, fields, timestamp)

```

When you create an instance of a custom BYOI plug-in, a JavaScript Object Notation (JSON) configuration file is attached inside the Kubernetes container for the BYOI ingest instance. The JSON configuration file contains information such as the device, device group, sensor path, hostname, and port of the backend service to which ingest data is sent. You can go through the JSON configuration using `/etc/byoi/config.json` for all the available attributes.

[Table 11 on page 94](#) lists several key attributes in the JSON configuration file.

Table 11: Attributes in the JSON Configuration File

Attributes	Description	How to Access Attributes
tand_host	Host name of the backend service to which the plug-in sends the ingest data.	Environment Variable <code>\$TAND_HOST</code>
tand_port	Port number of the backend service to which the plug-in sends the ingest data.	Environment Variable <code>\$TAND_PORT</code>

Table 11: Attributes in the JSON Configuration File (Continued)

Attributes	Description	How to Access Attributes
database	<p>Name of database where the ingest data is stored.</p> <p>The value for this attribute differs for each Paragon Insights node.</p>	<pre>config_json['hbin'] ['inputs']['plugin']['config'] ['device']['idx']['healthbot-storage'] ['database']</pre>
measurement	<p>Measurement of database in line protocol. See the InfluxDB documentation to learn more about measurement.</p> <p>For example, <i>topic/rule/sensor_name/byoi</i>.</p> <p>The value of sensor_name differs from sensor to sensor.</p>	<pre>config_json['hbin']['inputs']['plugin'] ['config']['device']['idx']['sensor'] [sensor_idx]['measurement']</pre>
fields	<p>Metric-value pairs, separated by comma without space.</p> <p>For example, cpu_usage=50,memory_utilization=12.</p>	None
timestamp	Unix Epoch timestamp in nanoseconds.	None
password	<p>Encoded password of the device that receives the streaming data.</p> <p>See "Decode Device Password" on page 95 for decoding device password.</p>	<pre>config_json['hbin'] ['inputs']['plugin']['config'] ['device']['idx']['authentication'] ['password']['password']</pre>

Decode Device Password

The JSON configuration file can contain encoded sensitive information such as the password of the device that streams data.

To decode the data, you can initiate a POST call using the API `api-server:9000/api/v2/junos-decode` inside the plugin container, with the encoded data in the post body.

The following sample POST call decodes an encoded password present in the JSON configuration file.

```
curl -X POST -L api-server:9000/api/v2/junos-decode -H "Content-Type: application/json" -d '{"data": "$ABC123"}' -v
```

Create a Shell Script for Configuration Updates

When the BYOI ingest image configuration or the Paragon Insights device group configuration changes, the JSON configuration file is updated. When there is a change in configuration, Paragon Insights must re-read the configuration file by invoking the shell script located at `/jfit_scripts/jfit_reconfigure.sh`.

When you build the ingest plugin image, you can name your shell script `jfit_reconfigure.sh` and copy the script to `/jfit_scripts/` folder.

In the shell script, you can send a `SIGHUP` signal to the main process or simply kill old processes and start new ones. The following example shell script sends a `SIGHUP` signal to the main plug-in process:

```
pid=`ps -ef | grep ".*main.py" | grep -v 'grep' | awk '{ print $1}'` && \
kill -s HUP $pid
```

Tag and Export the BYOI Custom Plugin Image

After you build the custom plugin, you must tag the plug-in image and export it as a tar file. You can tag the plug-in image in the `healthbot_plugin_name:your_version` format. The plug-in image must be exported as a compressed tar file using the following command:

```
docker save tag -o healthbot_plugin_name.tar.gz
```

Configure Kubernetes YAML File

The Kubernetes Jinja template file has the basic configuration required to deploy Kubernetes resources such as containers for the for the BYOI ingest pod.

You can use the following sample Kubernetes Jinja template to create a YAML file. You must replace:

- Placeholders for commands and args, `<ADD_COMMAND>` and `<ADD_ARGUMENTS>`. For example, replace `<ADD_COMMAND>` with `python3` and `<ADD_ARGUMENTS>` with the name of your Python file.
- `<PLUGIN_NAME_CAPITALIZED>` with your plugin name in uppercase letters.

You can add other properties to the 'containers' part, such as volumes or Kubernetes secrets, in the template. After you modify the sample Kubernetes Jinja template, change the name of the file to `healthbot_<plugin_name>.yaml.j2` and save it.

Sample Kubernetes Jinja template

```

set sg_name = '-' + env['SUBGROUP'] -%}
{%- set sg_dir = '_' + env['SUBGROUP'] -%}
{% if env['SUBGROUP'] == '' -%}
  {%- set sg_name = '' -%}
  {%- set sg_dir = '' -%}
{%- endif %}
kind: ConfigMap
apiVersion: v1
metadata:
  namespace: {{ env['NAMESPACE'] }}
  name: {{ env['GROUP_TYPE'] }}-{{ env['GROUP_NAME_VALID'] }}
    {{ sg_name }}-{{ env['CUSTOM_PLUGIN_NAME'] }}
  labels:
    app: {{ env['CUSTOM_PLUGIN_NAME'] }}
    group-name: {{ env['GROUP_NAME'] }}
    group-type: {{ env['GROUP_TYPE'] }}
    subgroup: {{ env['SUBGROUP'] }}
data:
  TAND_HOST: '{{ env['GROUP_TYPE_SHORT'] }}-{{ env['GROUP_NAME_VALID'] }}
    {{ sg_name }}-{{ env['CUSTOM_PLUGIN_NAME'] }}-terminus'
  TAND_PORT: '{{env['tand:TAND_PORT']}}'
  PUBLISHD_HOST: '{{env['publishd:PUBLISHD_HOST']}}'
  PUBLISHD_PORT: '{{env['publishd:PUBLISHD_PORT']}}'
  CONFIG_MANAGER_PORT: {{env['configmanager:CONFIG_MANAGER_PORT']}}
  CHANNEL: '{{ env['GROUP_TYPE'] }}-{{ env['GROUP_NAME'] }}'
  GODEBUG: 'madvdontneed=1'
  IAM_SERVER: '{{ env['iam:IAM_SERVER'] }}'
  IAM_SERVER_PORT: '{{ env['iam:IAM_SERVER_PORT'] }}'
  IAM_SERVER_PROTOCOL: '{{ env['iam:IAM_SERVER_PROTOCOL'] }}'
  IAM_NAMESPACE: '{{ env['iam:IAM_NAMESPACE'] }}'

```

```

---
apiVersion: apps/v1
kind: Deployment
metadata:
  namespace: {{ env['NAMESPACE'] }}
  name: {{ env['GROUP_TYPE'] }}-{{ env['GROUP_NAME_VALID'] }}
    {{ sg_name }}-{{ env['CUSTOM_PLUGIN_NAME'] }}
  labels:
    app: {{ env['CUSTOM_PLUGIN_NAME'] }}
    group-name: {{ env['GROUP_NAME'] }}
    group-type: {{ env['GROUP_TYPE'] }}
    subgroup: {{ env['SUBGROUP'] }}
spec:
  replicas: 1
  selector:
    matchLabels:
      app: {{ env['CUSTOM_PLUGIN_NAME'] }}
      group-name: {{ env['GROUP_NAME'] }}
      group-type: {{ env['GROUP_TYPE'] }}
      subgroup: {{ env['SUBGROUP'] }}
  template:
    metadata:
      namespace: {{ env['NAMESPACE'] }}
      labels:
        app: {{ env['CUSTOM_PLUGIN_NAME'] }}
        group-name: {{ env['GROUP_NAME'] }}
        group-type: {{ env['GROUP_TYPE'] }}
        subgroup: {{ env['SUBGROUP'] }}
    spec:
      tolerations:
        - key: "node.kubernetes.io/not-ready"
          operator: "Exists"
          effect: "NoExecute"
          tolerationSeconds: 180
        - key: "node.kubernetes.io/unreachable"
          operator: "Exists"
          effect: "NoExecute"
          tolerationSeconds: 180
      initContainers:
        - name: sync
          image: {{ env['REGISTRY'] }}/{{ env['HEALTHBOT_INIT_CONTAINER_IMAGE'] }}:
            {{ env['HEALTHBOT_INIT_CONTAINER_TAG'] }}

```



```

imagePullPolicy: Always
command: ["python3"]
args: ["/root/sync_files.py", "-c", "{{ env['GROUP_TYPE'] }}-
      {{ env['GROUP_NAME'] }}" ]
env:
- name: NODE_IP
  valueFrom:
    fieldRef:
      fieldPath: status.hostIP
envFrom:
- configMapRef:
    name: {{ env['GROUP_TYPE'] }}-{{ env['GROUP_NAME_VALID'] }}
      {{ sg_name }}-{{ env['CUSTOM_PLUGIN_NAME'] }}
containers:
- name: {{ env['CUSTOM_PLUGIN_NAME'] }}
  image: {{ env['REGISTRY'] }}/{{ env['HEALTHBOT_<PLUGIN_NAME_CAPITALIZED>_IMAGE'] }}:
    {{ env['HEALTHBOT_<PLUGIN_NAME_CAPITALIZED>_TAG'] }}
  imagePullPolicy: Always
  #example
  #command: ["python3"]
  #args: ["/main.py"]
  command: [<ADD_COMMAND>]
  args: [<ADD_ARGUMENTS>]
  env:
- name: NODE_IP
  valueFrom:
    fieldRef:
      fieldPath: status.hostIP
envFrom:
- configMapRef:
    name: {{ env['GROUP_TYPE'] }}-{{ env['GROUP_NAME_VALID'] }}
      {{ sg_name }}-{{ env['CUSTOM_PLUGIN_NAME'] }}
volumeMounts:
- name: default
  mountPath: /etc/byoi
- name: data-model
  mountPath: /etc/ml
volumes:
- name: default
  hostPath:
    type: DirectoryOrCreate
    path: {{ env['JFIT_OUTPUT_PATH'] }}/{{ env['GROUP_NAME'] }}
      {{ sg_dir }}/custom_{{ env['CUSTOM_PLUGIN_NAME'] }}_collector

```

```

- name: data-model
  hostPath:
    type: DirectoryOrCreate
    path: {{ env['JFIT_ETC_PATH'] }}/data/models/{{ env['GROUP_NAME'] }}
  imagePullSecrets:
    - name: registry-secret

```

(Optional) Assign Virtual IP Address to Plugin

For a custom BYOI plug-in to be reachable from an external network, the plug-in needs to be exposed as a Kubernetes load balancer service. This is an optional configuration. By default, the plug-in uses virtual IP address of the Paragon Insights gateway. You can also assign a custom virtual IP address and add the following template to the end of the Kubernetes Jinja template file in ["Configure Kubernetes YAML File" on page 96](#).

Ensure that you replace `<PLUGIN_PORT>` and `<PROTOCOL>` in the given template with your desired values such as port 80 for protocol HTTP. See [Kubernetes documentation](#) for supported protocols.

To configure a custom virtual IP for the BYOI plugin, replace `<custom_load_balancer_ip>` with an IP address.

```

---
{% set service_values = env.get('SERVICE_VALUES', {}) -%}
{%- set global_annotations = service_values.get('annotations') -%}
{%- set global_load_balancer_ip = service_values.get('loadBalancerIP') -%}
{%- set custom_annotations = service_values.get(svc_name, {}).get('annotations') -%}
{%- set custom_load_balancer_ip = service_values.get(svc_name, {}).get('loadBalancerIP') -%}
{%- set service_type = service_values.get(svc_name, {}).get('type', 'LoadBalancer') -%}
{%- for ip in env['LOAD_BALANCER_IPS'] %}
apiVersion: v1
kind: Service
metadata:
  namespace: {{ env['NAMESPACE'] }}
  {%- if loop.index0 == 0 %}
  name: {{ env['GROUP_TYPE_SHORT'] }}-{{ env['GROUP_NAME_VALID'] }}
        {{ sg_name }}-{{ env['CUSTOM_PLUGIN_NAME'] }}
  {%- else %}
  name: {{ env['GROUP_TYPE_SHORT'] }}-{{ env['GROUP_NAME_VALID'] }}

```

```

        {{ sg_name }}-{{ env['CUSTOM_PLUGIN_NAME'] }}-{{loop.index0}}
    {% endif %}
    labels:
      app: {{ env['CUSTOM_PLUGIN_NAME'] }}
      group-name: {{ env['GROUP_NAME'] }}
      group-type: {{ env['GROUP_TYPE'] }}
      subgroup: '{{ env['SUBGROUP'] }}'
    annotations:
      {% if env.get('LOADBALANCER_PROVIDER', 'User') == 'HealthBot' %}
        metallb.universe.tf/allow-shared-ip: healthbot-{{loop.index0}}
      {% elif custom_annotations %}
        {{ custom_annotations }}
      {% elif global_annotations %}
        {{ global_annotations }}
      {% else %}
        {}
      {% endif %}
    spec:
      type: LoadBalancer
      {%- if env.get('LOADBALANCER_PROVIDER', 'User') == 'HealthBot' %}
        loadBalancerIP: {{ ip }}
      {%- elif custom_load_balancer_ip %}
        loadBalancerIP: {{ custom_load_balancer_ip }}
      {%- elif global_load_balancer_ip %}
        loadBalancerIP: {{ global_load_balancer_ip }}
      {%- endif %}
      selector:
        app: {{ env['CUSTOM_PLUGIN_NAME'] }}
        group-name: {{ env['GROUP_NAME'] }}
        group-type: {{ env['GROUP_TYPE'] }}
        subgroup: '{{ env['SUBGROUP'] }}'
      ports:
        - name: port
          port:<PLUGIN_PORT>
          protocol:<PROTOCOL>
    {% endfor %}

```

After you configure the port and protocol, external applications can communicate with the custom BYOI plug-in via `<gateway_IP>:<PLUGIN_PORT>`. If you configured a custom virtual IP for the plug-in to act as a load balancer service, external applications can communicate with the plugin via `custom_load_balancer_ip:PLUGIN_PORT`.

Use `0.0.0.0:<PLUGIN_PORT>` to connect to the server running inside the plug-in from the Kubernetes host.

You can configure different ports for different applications in the given Kubernetes Jinja template under the ports section.

NOTE: If you configure different port numbers in the Kubernetes Jinja template, you must hard code the port number and the corresponding port name in your respective applications.

Load the BYOI Custom Plugin

Mount the BYOI custom plug-in image tar file and the modified Kubernetes YAML file to the Paragon Insights primary node and load the plug-in in the Paragon Insights management CLI.

1. Mount the BYOI plugin image (compressed tar file) using the following command:

```
export HB_EXTRA_MOUNT1=/path/to/healthbot_<plugin_name>.tar.gz
```

2. Mount the Kubernetes YAML file using the following command:

```
export HB_EXTRA_MOUNT2=/path/to/healthbot_<plugin_name>.yaml.j2
```

3. Load the plugin image and the Kubernetes YAML file using the following command:

```
sudo -E healthbot load-plugin -i $HB_EXTRA_MOUNT1 -c $HB_EXTRA_MOUNT2
```

You can see a confirmation message when the plugin loads successfully.

4. (Optional) Select the **Configuration > Data Ingest > Settings > BYO Ingest Plugins** page and view the custom plug-in in the **Custom Plugins** tab.

After you load your plug-in, create an instance of the custom BYOI plug-in in the Bring Your Own Ingest page. Since custom plug-ins do not use the default Paragon Automation resources, you must configure a new rule and a playbook for the ingest plug-in.

Configure Bring Your Own Ingest Custom Plug-in Instances

For custom ingest plug-ins, you must configure key-value pairs in the Paragon Insights GUI.

To configure a custom ingest plug-in instance:

1. Select **Configuration > Ingest > BYOI Plugins**.

The Bring Your Own Ingest Plugin page appears.

2. Click **+New Instance** in the Custom tab.

The Create Custom Plugins page appears.

3. Enter the details as described in [Table 12 on page 103](#).

4. Do one of the following:

- **Save**—Save your ingest mapping but do not deploy the updated configuration. You can use this option when, for example, you are making several changes and want to deploy all your updates at the same time later. See *Commit or Roll Back Configuration Changes in Paragon Insights* for more information.
- **Save & Deploy**—Save the custom plug-in instance configuration and deploy the configuration in Paragon Insights.

Fields marked with an asterisk (*) are mandatory.

Table 12: Fields on the Create Custom Plugins Page

Fields	Description
Name	Enter the name of the custom ingest plug-in instance.
Plugin Name	Enter the name given to the input ingest plug-in that you created.
Service Name	Enter a name for the service. The service name can be same as the name of the custom ingest plug-in.
Key	Name of key parameters in the ingest plug-in. For example, sensor or frequency. You can add one or more ingest parameters as a key using the + icon.
Value	Enter the value for the key parameter. For example, you must enter a sensor path as value if sensor is your key.
Authentication	
SASL Username	Enter the plain text username for authentication.

Table 12: Fields on the Create Custom Plugins Page (*Continued*)

Fields	Description
Password	Enter a password. The password must be 6 to 128 characters long and must contain a combination of uppercase and lowercase characters. It must also contain numbers and special characters.
TLS	
CA Profile Name	Enter the name of the trusted certificate authority (CA).
Local Certification Profile Name	Enter the name of the self-signed certificate you created for Paragon Insights.
Skip Certification Chain and Host Verification	Toggle the switch on if you want to skip verifying the integrity of the CA certificate. NOTE: Connection between the input ingest plug-in and the external data source is encrypted even if you skip verification of the certificate.

After you configure a custom plug-in instance, you must create a new rule with the BYOI ingest parameters. To collect data using the custom ingest plug-in, you must add this rule to a new playbook and run a playbook instance on device or network groups. See ["Use the Sample Rule and Playbook Configurations for BYOI Custom Plug-ins" on page 105](#) for sample rule and playbook configurations for custom BYOI plugins.

RELATED DOCUMENTATION

| [Understand Bring Your Own Ingest](#) | 81

Use the Sample Rule and Playbook Configurations for BYOI Custom Plug-ins

Before you configure a custom rule, playbook, and device group for the Bring Your Own Ingest (BYOI) custom plug-in, you must create an instance of the custom plug-in. See ["Configure Bring Your Own Ingest Custom Plug-in Instances" on page 102](#) for more information.

You must write your own rules for custom Bring Your Own Ingest plug-ins. You must deploy a custom rule in a new playbook that you create for the BYOI plug-in ingest. You must also create a new device group that you add later in the BYOI playbook. Enter the rule and playbook configurations in Paragon Insights management CLI.

The following sample rule triggers an alert when value of *key* is greater than 5. The trigger alert frequency is set to 10 seconds.

```
set healthbot topic external rule r1 sensor sensor_a byoi plugin name example-plug-in
set healthbot topic external rule r1 field key sensor sensor_a path key
set healthbot topic external rule r1 field key type integer
set healthbot topic external rule r1 trigger trigger_789 frequency 10s
set healthbot topic external rule r1 trigger trigger_789 term Term_1 when greater-than "$key" 5
set healthbot topic external rule r1 trigger trigger_789 term Term_1 then status color red
set healthbot topic external rule r1 trigger trigger_789 term Term_1 then status message BAD
set healthbot topic external rule r1 trigger trigger_789 disable-alarm-deduplication
```

NOTE:

- The *example-plug-in* in the sample rule refers to the plug-in name you entered in the **Name** field, when creating an instance of the custom plug-in.
- The *sensor_a* in the sample rule refers to the value configured for the key parameter in the **Value** field, when creating an instance of the custom plug-in.

Use the following sample command to create a playbook p1 for the BYOI ingest that uses the sample rule r1 that you previously configured.

```
set healthbot playbook p1 rules external/r1
```

After you deploy the playbook configuration, you must create a new instance of the playbook in the Playbooks page. Then, you can also monitor the health of the BYOI ingest in the Health page (**Monitoring > Health**).

RELATED DOCUMENTATION

[Understand Bring Your Own Ingest | 81](#)

Delete a Bring Your Own Ingest Plug-in

To delete a Bring Your Own Ingest plug-in:

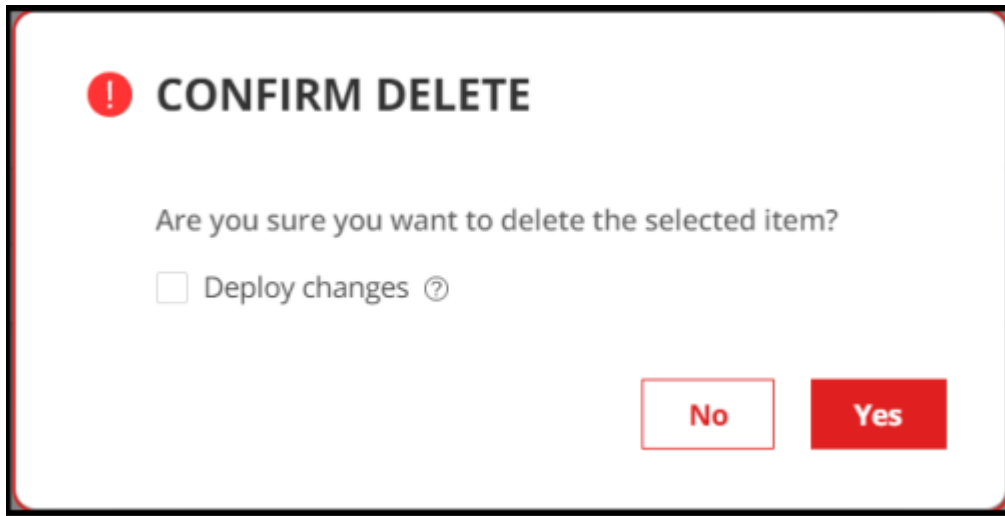
1. Click **Settings > Ingest** from the left-nav bar.
The **Ingest Settings** page is displayed.
2. Click the **BYO Ingest Plugins** tab to view the **Bring Your Own Ingest** page.
3. Do one of the following.

NOTE: Default plug-ins cannot be deleted.

- To delete an ingest mapping:
 - a. Click **Ingest Mapping**.
 - b. Select the ingest mapping that you want to delete.
 - c. Click the **delete (trash can)** icon.
- To delete a custom plug-in:
 - a. Click **Custom Plugins**.
 - b. Select the custom plug-in that you want to delete.
 - c. Click the **delete (trash can)** icon.

The **CONFIRM DELETE** pop-up appears.

Figure 12: Confirm Delete Pop-up



4. Do any one of the following:

- Click **Yes** to delete Bring Your Own Ingest plug-in information from the database. However, the changes are not applied to the ingest service.

NOTE:

- We recommended that you do not delete a Bring Your Own Ingest plug-in that is currently in use.
 - After you delete a Bring Your Own Ingest plug-in from the database, you cannot map that plug-in with another device group even if you have not deployed changes.
 - You can also deploy changes to the ingest service or roll back the changes that you have already deleted, from the **PENDING CONFIGURATION** page. For more information, see *Commit or Roll Back Configuration Changes in Paragon Insights*.
- Select the **Deploy changes** check box and then click **Yes** to delete Bring Your Own Ingest plug-in information from the database, and to apply the changes to the ingest service.
 - (Optional) Click **No** to cancel this operation.

2

CHAPTER

“Pull” Model Data Ingest Methods

[Paragon Insights Pull-Model Overview | 109](#)

[Paragon Insights Pull-Model Ingest Methods | 109](#)

Paragon Insights Pull-Model Overview

While the 'push' model is the preferred approach for its efficiency and scalability, there are still cases where the 'pull' data collection model is appropriate. Two examples might be when a device doesn't support the Junos Telemetry Interface (JTI), or when managing third party devices. With the pull model, Paragon Insights requests data from network devices at periodic, user-defined intervals.

Paragon Insights Pull-Model Ingest Methods

IN THIS SECTION

- [Server Monitoring Ingest | 109](#)
- [Understanding kube-state-metrics Service | 116](#)
- [iAgent \(CLI/NETCONF\) | 128](#)
- [SNMP | 142](#)

Paragon Insights currently supports the following pull-model sensors:

- ["Server Monitoring Ingest" on page 109](#)
- [Understanding kube-state-metrics Service](#)
- ["iAgent \(CLI/NETCONF\)" on page 128](#)
- ["SNMP" on page 142](#)

Server Monitoring Ingest

IN THIS SECTION

- [Configure a Rule Using Server Monitoring Sensor | 114](#)

Starting with Paragon Insights Release 4.1.0, the Server Monitoring sensor collects data from servers and virtual machines on which you host the Paragon application. The sensor uses the third-party plug-in, Node Exporter. The Node Exporter plug-in is pre-installed in all server clusters of Paragon Insights. In the GUI, the default servers and virtual machines deployed in the Paragon Insights cluster are represented as devices that are automatically added to the **Paragon-Cluster** device group. The sensor collects data from servers and virtual machines to track CPU, memory, network, traffic, disk, and filesystem metrics. It writes the output to a time series database.

Paragon Insights has the following pre-configured playbooks to monitor server data.

- CPU utilization
- Disk reads and writes
- Errors, available bytes, and utilized bytes in filesystem
- Utilized bytes and available bytes in memory
- Received and transmitted total packet size, errors in received and transmitted packets, total received and transmitted multicast packets in network

When you configure a rule using Server Monitoring ingest, you can use some of the sensor paths listed in [Table 13 on page 110](#).

Table 13: Server Metrics

Sensor Path	Description
<code>/node/boot/time/seconds</code>	Boot time in each server node.
<code>/node/cpu/seconds/total</code>	The total time (in seconds) the CPU stays in idle, system, user, and nice modes.
<code>/node/disk/read/bytes/total</code>	The total number of bytes read successfully.
<code>/node/disk/read/errors/total</code>	The total number of read errors in nodes.
<code>/node/disk/read/retries/total</code>	The number of times the ingest tries to read from the disk if there is a failure.
<code>/node/disk/read/sectors/total</code>	The total number of sectors read successfully.

Table 13: Server Metrics (Continued)

Sensor Path	Description
<code>/node/disk/read/time/seconds/total</code>	The total time taken to complete reads successfully per node.
<code>/node/disk/reads/completed/total</code>	The total number of reads completed successfully.
<code>/node/disk/write/errors/total</code>	The total number of errors in writes.
<code>/node/disk/write/retries/total</code>	The number of times the ingest tries to write on the disk if there is a failure.
<code>/node/disk/write/time/seconds/total</code>	The total time taken to complete all writes.
<code>/node/disk/writes/completed/total</code>	The total number of writes completed per node.
<code>/node/disk/written/bytes/total</code>	The total number of bytes written successfully.
<code>/node/disk/written/sectors/total</code>	The total number of sectors written successfully.
<code>/node/exporter/build/info</code>	A metric that has the value '1' and has version, revision, go version, and branch from which node exporter is built.
<code>/node/filesystem/avail/bytes</code>	The filesystem size available to non-root users.
<code>/node/filesystem/device/error</code>	The number of I/O errors that occur when collecting data from a filesystem.
<code>/node/filesystem/files</code>	The total number of index nodes permitted in a node.
<code>/node/filesystem/files/free</code>	The number of index nodes that are free for use in a node.
<code>/node/filesystem/free/bytes</code>	The free space (in bytes) available for the user, excluding reserved blocks.

Table 13: Server Metrics (Continued)

Sensor Path	Description
<code>/node/filesystem/readonly</code>	Data that shows if the filesystem in a node is mounted as read-only.
<code>/node/filesystem/size/bytes</code>	The size of all files in bytes.
<code>/node/load1</code>	Load on each server/host node captured every 1 minute.
<code>/node/load15</code>	Load on each server/host node captured every 15 minutes.
<code>/node/load5</code>	Load on each server/host node captured every 5 minutes.
<code>/node/memory/active/bytes</code>	Size of memory in bytes that are actively used by processes.
<code>/node/memory/compressed/bytes</code>	Total size of compressed memory.
<code>/node/memory/free/bytes</code>	Total memory in bytes that is free for use in a node.
<code>/node/memory/inactive/bytes</code>	Memory bytes that are not actively used by processes.
<code>/node/memory/swap/total/bytes</code>	Total size of memory swapped in all nodes.
<code>/node/memory/swap/used/bytes</code>	The size of swapped memory used by nodes.
<code>/node/memory/swapped/in/bytes/total</code>	Size of memory swapped back to RAM in all nodes.
<code>/node/memory/swapped/out/bytes/total</code>	Size of memory swapped out from RAM in all nodes.
<code>/node/memory/total/bytes</code>	Total bytes of memory in all nodes.
<code>/node/memory/wired/bytes</code>	Memory that cannot be swapped out.

Table 13: Server Metrics *(Continued)*

Sensor Path	Description
<code>/node/network/receive/bytes/total</code>	Total size of packets received by a device.
<code>/node/network/receive/errs/total</code>	Total number of errors encountered by a device when receiving packets.
<code>/node/network/receive/multicast/total</code>	Total number of multicast packets received by a device.
<code>/node/network/receive/packets/total</code>	Total number of packets received by a device.
<code>/node/network/transmit/bytes/total</code>	Total size of packets received by a device.
<code>/node/network/transmit/errs/total</code>	Total number of errors encountered by a device when receiving packets.
<code>/node/network/transmit/multicast/total</code>	Total number of multicast packets transmitted by a device.
<code>node/network/transmit/packets/total</code>	Total number of packets transmitted by a device.
<code>/node/scrape/collector/duration/seconds</code>	Time taken by each collector to scrape metrics.
<code>/node/scrape/collector/success</code>	Number of times Node Exporter collector successfully scraped targets.
<code>/node/textfile/scrape/error</code>	Errors encountered by Node Exporter when scraping targets using textfile scripts.
<code>/node/time/seconds</code>	Displays system time in seconds in the node since epoch (1970).
<code>/node/uname/info</code>	Name of the node from which Node Exporter collects metrics.

The following tags such as mode, device etc. can be used as key fields applicable to all metrics listed under main metrics (`/node/cpu` or `/node/disk`). When you configure a key field in a rule, you can mention only the key field name in **Path** field.

- **/node/cpu/**
 - **cpu:** The number of cores available in CPU.
 - **mode:** The type of CPU usage in a node such as idle, system, user, and nice.
- **/node/disk/**
 - **device:** Name of disks such as disk0, disk1, sda, sdb, or sdc.
- **/node/filesystem/**
 - **device:** Disk paths such as /dev/sda1, /dev/sda2, and /dev/sdb1
 - **fstype:** Type of partition formatting such as ext4, NTFS (New Technology File System), and APFS (Apple File System).
 - **mountpoint:** Mount paths in the device.
- **/node/network/**
 - **device:** Interface names of the device such as wlan0, en0, or docker0.

Configure a Rule Using Server Monitoring Sensor

In the following example, you can use server monitoring sensor to collect disk read data from servers. You can configure fields for total disk read size, time taken to perform the reads, and name of the device that has the disk (key field). You can also calculate rate of disk read and configure a trigger alert when the total disk read exceeds a preset threshold.

1. Click **Configuration > Rules**.
2. On the Rules page, click **+ Add Rule**.
3. Enter topic name as server.monitoring and set the rule name after the slash (/).

In this example, rule name can be check-disk-read.

The rule name in the top row of the rule page follows the topic/rule name format. The default topic name is 'external' when you add a new rule.

4. Add a description and synopsis for your rule.
5. Click **+ Add Sensor** and enter a name for the sensor. For example, disk.
6. Select **Server Monitoring** as sensor type.
7. Enter sensor path as **/node/disk**.

If you add a / at the end of the path, you get sensor paths for disk reads, writes, and written records.

8. Enter a Frequency (in seconds) for the sensor. For example, 30s.

The minimum usable sensor frequency is 15 seconds. It takes at least 15 seconds before you see data from the ingest.

9. Go to Fields tab and click **+ Add Field** and enter the Field Name as **device-name**.
This field collects the name of the device containing the disk that generates read data.
10. Select Field Type as **string**.
11. Enable **Add to Rule Key**.
12. Select Ingest Type (Field Source) as **Sensor**.
13. Select the name of the sensor in the Sensor field. In this example, select **disk**.
14. Type **Device** in the Path field.
15. Go to Fields tab and click **+ Add Field** and enter the Field Name as **disk-read-total**.
This field collects the total size of disk reads in bytes.
16. Select Field Type as **float** and Ingest Type (Field Source) as **Sensor**.
17. Select the name of the sensor for the Sensor field. In this example, select **disk**
18. Select Path as **/node/disk/read/bytes/total**.
19. Click **+ Add Field** and enter the Field Name as **disk-read-rate**.
20. Select Field Type as **float** and Ingest Type (Field Source) as **Formula**.
21. In Formula, select **Rate of Change**.
22. In Field, select the field name **disk-read-total**.
23. Click **+ Add Field** and enter the Field Name as **read-threshold**.
This field contains a constant value for disk read threshold.
24. Select Field Type as **float** and Ingest Type (Field Source) as **Constant**.
25. In Constant Value, enter a threshold value for disk reads. For example, **5**.
26. Go to Triggers tab and click **+Add Trigger**.
27. Enter a name for the trigger such as **read-trigger**.
28. Enter Frequency. For example, **2o** (2 offset).
If you set frequency as 2o or 2 offset, it multiplies the static frequency you set for sensor frequency by 2.
29. Click **+Add Term** and enter a term name. For example, **high-disk-usage**.
30. In the When statement, select left operand as **disk-read-total** field, right operand as **read-threshold** field, and the operator as *Increase At Least by Value*.
31. In the Then statement, set red as the color and enter Message as **high disk read value**.
32. Do one of the following:
 - Click **Save** to save the rule configuration. The rule configuration is not deployed.
 - Click **Save and Deploy** to deploy the configuration in Paragon Insights Platform.

To collect data on metrics, you must add the rule to a Playbook and apply a Playbook instance to device or network groups.

When you start collecting server metrics, you can see the logs by providing the pod IDs for the ingest.

- Log in to the Paragon Insights management CLI.
- Type the command `healthbot k logs server-monitoring pod id`.

Understanding kube-state-metrics Service

IN THIS SECTION

- [Enable kube-state-metrics | 117](#)
- [Sample Rules and Playbooks | 117](#)

kube-state-metrics service is a Beta feature as of Paragon Insights Release 4.2.0.

kube-state-metrics is a third-party metrics monitoring service that generates metrics based on the current state of Kubernetes clusters. You can use kube-state-metrics to monitor the health of Kubernetes cluster and services. With Release 4.2.0, kube-state-metrics service is supported as a beta-only feature. kube-state-metrics runs as a cluster service, and is installed automatically when you install Paragon Insights. Once this service is installed, you can enable this service to generate, monitor, and expose metrics of various objects within a Kubernetes cluster.

kube-state-metrics service provides metrics on pods, DaemonSets, deployments, persistent volume, endpoints, ingress, job, lease, and configmap objects that are part of a Kubernetes cluster.

The following is a list of some metrics that are exposed:

- Pods running in a namespace
- Pods that are available
- Information on successful/failed deployments
- State of persistent volumes
- Information on currently running, successful, and failed jobs
- Pods that are in error state
- Health of deployment and DaemonSets
- Status and condition of Kubernetes nodes

Enable kube-state-metrics

You can enable kube-state-metrics from the CLI as well as from the Paragon Insights UI.

The following is an overview of steps to enable kube-state-metrics from the UI:

1. Create a device.

The device represents the cluster. The device hostname must be the kube-state-metrics service IP. For example, kube-state-metrics.healthbot.svc.cluster.local.

2. Add the device to a device group.

3. Create rules with server-monitoring sensor type and relevant kube-state-metrics sensor path(s).

4. Apply playbooks.

Sample Rules and Playbooks

check-daemonset-status.rule

```
healthbot {
  topic kube-metrics {
    rule check-daemonset-status {
      keys [ daemonset namespace ];
      synopsis "";
      description "Checks daemon set unavailable status";
      sensor daemonset-status {
        description "Checks daemon set unavailable status";
        server-monitoring {
          sensor-name /kube/daemonset;
          frequency 60s;
        }
      }
    }
    field daemonset {
      sensor daemonset-status {
        path daemonset;
      }
      type string;
      description "Checks status for demonset key";
    }
    field daemonset_status {
      sensor daemonset-status {
        path /kube/daemonset/status/number/unavailable;
      }
    }
  }
}
```



```

rule check-deployment-status-condition {
  keys [ condition deployment namespace ];
  synopsis "";
  description "Checks kube metrics deployment status condition";
  sensor deployment-status {
    description "Checks kube metrics deployment status condition";
    server-monitoring {
      sensor-name /kube/deployment;
      frequency 60s;
    }
  }
  field condition {
    sensor deployment-status {
      path condition;
    }
    type string;
    description "Deployment condition";
  }
  field deployment {
    sensor deployment-status {
      path deployment;
    }
    type string;
    description "Deployment pod name";
  }
  field namespace {
    sensor deployment-status {
      path namespace;
    }
    type string;
    description "Deployment namespace";
  }
  field status {
    sensor deployment-status {
      path status;
    }
    type string;
    description "Checks for true or false condition";
  }
  trigger deployment-status {
    frequency 1offset;
    term available {
      when {

```



```

        description "Deployment pod name";
    }
    field deployment_status {
        sensor deployment-status {
            path /kube/deployment/status/replicas;
        }
        type float;
        description "Field to check 0 or other values";
    }
    field namespace {
        sensor deployment-status {
            path namespace;
        }
        type string;
        description "Namespace key";
    }
    trigger deployment-status {
        frequency 1offset;
        term available {
            when {
                not-equal-to "$deployment_status" 0;
            }
            then {
                status {
                    color green;
                    message "Deployment status for replicaset is $deployment_status for
$namespace $deployment";
                }
            }
        }
        term notavailable {
            then {
                status {
                    color red;
                    message "Deployment status for replicaset is $deployment_status for
$namespace $deployment";
                }
            }
        }
    }
}

```

```

    }
}

```

check-pod-container-restarts.rule

```

healthbot {
  topic kube-metrics {
    rule check-pod-container-restarts {
      keys [ container namespace pod uid ];
      synopsis "";
      description "Checks pod container status restarts";
      sensor pod-init-container-status {
        description "Checks pod container status restarts";
        server-monitoring {
          sensor-name /kube/pod/container;
          frequency 60s;
        }
      }
      field container {
        sensor pod-init-container-status {
          path container;
        }
        type string;
        description "container name";
      }
      field namespace {
        sensor pod-init-container-status {
          path namespace;
        }
        type string;
        description "namespace of the pod";
      }
      field pod {
        sensor pod-init-container-status {
          path pod;
        }
        type string;
        description "pod name";
      }
      field restart-value {
        sensor pod-init-container-status {
          path /kube/pod/container/status/restarts/total;

```



```

    }
    type integer;
    description "restart status value";
}
field uid {
    sensor pod-init-container-status {
        path uid;
    }
    type string;
    description "Id ";
}
trigger restart-total-status {
    frequency 1offset;
    term less-than-ten {
        when {
            less-than-or-equal-to "$restart-value" 10;
        }
        then {
            status {
                color green;
                message "$namespace pod $pod container $container restart total is
$restart-value ";
            }
        }
    }
    term between-ten-and-twenty {
        when {
            range "$restart-value" {
                min 10;
                max 20;
            }
        }
        then {
            status {
                color yellow;
                message "$namespace pod $pod container $container restart total is
$restart-value ";
            }
        }
    }
    term more-than-twenty {
        then {
            status {

```



```

        path pod;
    }
    type string;
    description "pod name";
}
field status {
    sensor pod-init-container-status {
        path /kube/pod/init/container/status/waiting;
    }
    type integer;
    description "Status value (0/1)";
}
field uid {
    sensor pod-init-container-status {
        path uid;
    }
    type string;
    description "Id ";
}
trigger waiting-status {
    frequency 1offset;
    term matches-zero {
        when {
            equal-to "$status" 0 {
                time-range 10offset;
            }
        }
        then {
            status {
                color green;
                message "$namespace $pod $container status is $status";
            }
        }
    }
    term is-not-zero {
        then {
            status {
                color red;
                message "$namespace $pod $container status is $status";
            }
        }
    }
}
}

```

```

    }
  }
}

```

check-node-status.rule

```

healthbot {
  topic kube-metrics {
    rule check-node-status {
      keys [ condition node ];
      synopsis "";
      description "Checks node status";
      sensor node-status {
        description "Checks node status";
        server-monitoring {
          sensor-name /kube/node;
          frequency 60s;
        }
      }
      field condition {
        sensor node-status {
          path condition;
        }
        type string;
        description "Node condition";
      }
      field node {
        sensor node-status {
          path node;
        }
        type string;
        description "Node name";
      }
      field node_status {
        constant {
          value "{{value}}";
        }
        type integer;
        description "Field to check condition";
      }
      field status {
        sensor node-status {

```



```
rules [ kube-metrics/check-daemonset-status kube-metrics/check-deployment-status-
replicas kube-metrics/check-node-status kube-metrics/check-pod-container-restarts kube-metrics/
check-pod-init-container-status kube-metrics/check-deployment-status-condition ];
description "Rules to check for kube-metrics ";
synopsis "Rules to check for kube-metrics ";
}
}
```

iAgent (CLI/NETCONF)

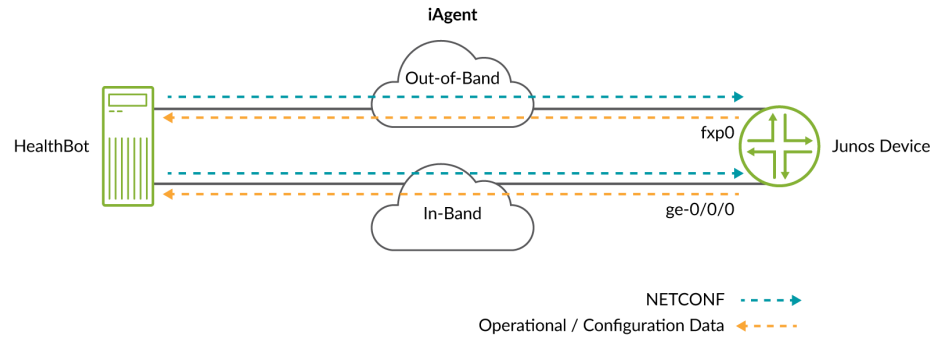
IN THIS SECTION

- [Example: PaloAlto Panos- Show Running Security Policy | 131](#)
- [Outbound SSH \(Device-Initiated\) | 135](#)
- [iAgent - vCenter/ESXi Server Monitor | 142](#)

For all the benefits of the 'push' data collection methods, some operational and state information is available only through CLI/VTY commands. iAgent fills this gap by taking advantage of NETCONF/SSH functionality to provide Paragon Insights with the ability to connect to a device, run commands, and capture the output.

iAgent sensors use NETCONF/SSH and YAML-based PyEZ tables and views to fetch the necessary data. Both structured (XML) and unstructured (VTY commands and CLI output) data is supported.

With iAgent, the Paragon Insights server initiates SSH requests over any available network interface, whether in-band or out-of band; and the device responds (when properly configured) with the requested data.



g300770

At minimum, iAgent (NETCONF) requires:

- Junos OS Version: 11.4 or later
- Minimum required device configuration:

```
set system services netconf ssh
```

Starting in HealthBot Release 3.1.0, iAgent functionality is extended to third party devices. When adding a device, you can choose **Other Vendor** from the **Vendor** pull-down. This adds the **Vendor Name** text field below the **Vendor** pull-down. Then you fill in the **iAgent Port Number**, **Vendor Name**, and **OS name** fields highlighted in [Figure 13 on page 130](#) to allow iAgent connections to non-Juniper devices.

NOTE: Refer to vendor documentation to understand how to configure third-party vendor devices to allow these connections.

Figure 13: Add Third-Party Device

Add Device(s)

Name

Device Name - defaults to hostname.

Hostname / IP address / Range*
edge0 / 10.4.4.112 / 10.4.4.[11-12]

Specify the host(s) to register.

Device Group(s)

Select device group(s) to include this device

System ID to use for JTI **Flow Source IPs**

Specify in the format - "Hostname:IP" used in JTI UDP messages. Comma separated values...

Initial Sync **gNMI Support** **gNMI Encoding**

Inherit from device-group Inherit from device-group Inherit from device-group

Select if initial sync to be performed for open-config sensors Select if gNMI support is enabled Select encoding to be used for gNMI

Open Config Port Number* **iAgent Port Number***

32767 22

Specify the Open Config Port Number. Specify the iAgent Port Number.

Vendor

Other Vendor

Select the device vendor.

Vendor Name **OS name**

PaloAlto Panos

SNMP Port Number **SNMP Community**

161 public

Specify the SNMP Agent Port. Enter the SNMP Agent Community.

Timezone **Syslog Source IPs** **Syslog Host Names**

+/-00:00 Comma separated values... Comma separated values...

Specify the Timezone in the format +/-hh:mm. Specify the IP addresses associated with this device in the syslog. Specify the host name associated with this device in the syslog.

Authentication

Tagging Profiles

CANCEL
SAVE
SAVE & DEPLOY

Using Netmiko, Paragon Insights makes persistent SSH connections over the selected port to the third-party device. To gather device information, Paragon Insights sends CLI commands over SSH and receives string blobs back as output. The string blobs are then parsed through TextFSM, using ntc-templates into JSON format and then stored in the database. Default templates are located at `/srv/salt/_textfsm`. A repository of ntc-templates for network devices is available here: [NTC Templates](#). For advanced users

who need a template which does not exist, you can create your own templates and upload them to Paragon Insights using the **Upload Rule Files** button on the **Configuration > Rules** page. User defined templates are stored at `/jft/_textfsm`. The files must end with the `.textfsm` suffix.

TextFSM is integrated into PyEZ's table/view feature which is an integral part of iAgent.

Example: PaloAlto Panos- Show Running Security Policy

IN THIS SECTION

- [Define PyEZ Table/View | 131](#)
- [Gather Output from Device | 132](#)
- [Generate JSON for Use in Paragon Insights Database | 133](#)

To see the running security policy on a Panos device, we need to:

- Define a table/view for it
- Gather the output by sending the needed CLI to the device over SSH
- Generate JSON to store in Paragon Insights database

Define PyEZ Table/View

We need to define a PyEZ table that is used by the iAgent rule assigned to the Panos device. The following table definition lacks a view definition. Because of this, the entire output from the `show running security-policy` ends up getting stored in the database after processing.

```
---
PanosSecurityPolicyTable:
  command: show running security-policy
  platform: paloalto_panos
  key: NAME
  use_textfsm: True
```

(Optional) To store only a portion of the received data in Paragon Insights, you can define a view in the same file. The view tells Paragon Insights which fields to pay attention to.

```

---
PanosSecurityPolicyTable:
  command: show running security-policy
  platform: paloalto_panos
  key: NAME
  use_textfsm: True
  view: TrafficAndActionView

TrafficAndActionView:
  fields:
    source: SOURCE
    destination: DESTINATION
    application_service: APPLICATION_SERVICE
    action: ACTION

```

Gather Output from Device

Using an iAgent rule that references the PyEZ table (or table/view) defined above, Paragon Insights sends the command `show running security-policy` to the device which produces the following output:

```

"intrazone-default; index: 1" {
  from any;
  source any;
  source-region none;
  to any;
  destination any;
  destination-region none;
  category any;
  application/service 0:any/any/any/any;
  action allow;
  icmp-unreachable: no
  terminal yes;
  type intrazone;
}

"interzone-default; index: 2" {
  from any;

```

```

source any;
source-region none;
to any;
destination any;
destination-region none;
category any;
application/service 0:any/any/any/any;
action deny;
icmp-unreachable: no
terminal yes;
type interzone;
}

dynamic url: no

```

Generate JSON for Use in Paragon Insights Database

Since the device configuration specifies Palo Alto Networks as the vendor and Panos OS as the operating system, the TextFSM template used for this example would look like this:

```

Value Key,Filldown NAME (.*)
Value Required FROM (\S+)
Value SOURCE (\S+)
Value SOURCE_REGION (\S+)
Value TO (\S+)
Value DESTINATION ([\S+\s+]*)
Value DESTINATION_REGION (\S+)
Value USER (\S+)
Value CATEGORY (\S+)
Value APPLICATION_SERVICE ([\S+\s+]*)
Value ACTION (\S+)
Value ICMP_UNREACHABLE (\S+)
Value TERMINAL (\S+)
Value TYPE (\S+)

Start
^${NAME}\s+\{
^\s+from\s+${FROM};
^\s+source\s+${SOURCE};
^\s+source-region\s+${SOURCE_REGION};
^\s+to\s+${TO};

```

```

^\s+destination\s+${DESTINATION};
^\s+destination-region\s+${DESTINATION_REGION};
^\s+user\s+${USER};
^\s+category\s+${CATEGORY};
^\s+application/service\s+${APPLICATION_SERVICE};
^\s+action\s+${ACTION};
^\s+icmp-unreachable:\s+${ICMP_UNREACHABLE}
^\s+terminal\s+${TERMINAL};
^\s+type\s+${TYPE};
^} -> Record

```

When the template above is used by Paragon Insights to parse the output shown previously, the resulting JSON looks like:

```

{"interzone-default; index: 2": {'ACTION': 'deny',
                                'APPLICATION_SERVICE': '0:any/any/any/any',
                                'CATEGORY': 'any',
                                'DESTINATION': 'any',
                                'DESTINATION_REGION': 'none',
                                'FROM': 'any',
                                'ICMP_UNREACHABLE': 'no',
                                'SOURCE': 'any',
                                'SOURCE_REGION': 'none',
                                'TERMINAL': 'yes',
                                'TO': 'any',
                                'TYPE': 'interzone',
                                'USER': ''},
 "intrazone-default; index: 1": {'ACTION': 'allow',
                                  'APPLICATION_SERVICE': '0:any/any/any/any',
                                  'CATEGORY': 'any',
                                  'DESTINATION': 'any',
                                  'DESTINATION_REGION': 'none',
                                  'FROM': 'any',
                                  'ICMP_UNREACHABLE': 'no',
                                  'SOURCE': 'any',
                                  'SOURCE_REGION': 'none',
                                  'TERMINAL': 'yes',
                                  'TO': 'any',
                                  'TYPE': 'intrazone',
                                  'USER': ''}}

```

Outbound SSH (Device-Initiated)

IN THIS SECTION

- [Configure TCP Port for Outbound SSH in Ingest | 140](#)
- [Connect a Device in Multiple Device Groups via Outbound SSH | 140](#)

Starting with Release 3.2.0, HealthBot can use iAgent connections that are device-initiated using outbound SSH. This configuration makes Paragon Insights act as the client to the device making the connection. This type of connection is useful in environments in which the remote devices cannot accept incoming connections. All existing iAgent rules can be used when outbound SSH is configured in Junos OS devices.

NOTE: In the 3.2.0 release, outbound SSH is only supported on Junos OS devices.

At minimum, iAgent (outbound-ssh) requires:

- Junos OS Version: 11.4 or later
- Minimum required device configuration:

```
set system services outbound-ssh client client_name device-id <device-name-in-healthbot>
set system services outbound-ssh client client_name keep-alive retry 30
set system services outbound-ssh client client_name keep-alive timeout 35
set system services outbound-ssh client client_name services netconf
set system services outbound-ssh client client_name 10.1x.x0.1 port 2222
```

NOTE: In the configuration above, *client_name* can be anything that makes sense. The IP address and port shown are examples representing Paragon Insights's IP address and a unique port number used for the devices in 1 device-group.

Outbound SSH is disabled by default and can be enabled at the device-group level. Once enabled, all devices in the group use outbound SSH unless specifically configured not to. Users can disable outbound SSH in a device through management CLI.

When you enable outbound SSH within the device-group, you must select a TCP port number over which all the member devices initiate their NETCONF connections to Paragon Insights. This port must be unique across the Paragon Insights installation. [Figure 14 on page 137](#) shows the **outbound SSH** section of the add/edit device group window.

Figure 14: Outbound-SSH - Device Group Level

outbound SSH ^

Port(s)
Comma separated values

Specify the outbound-ssh port(s).

CANCEL SAVE SAVE & DEPLOY

NOTE: To disable an individual device in a device-group from using outbound SSH, you must edit the device and select *disable* in the outbound SSH configuration section. If you later change your mind and want that device to use outbound SSH, edit the device and set outbound SSH to *reset*. [Figure 15 on page 139](#) is an edited (shortened) device add/edit window with the **outbound SSH** section shown.

Figure 15: Outbound-SSH - Device Level

Edit "spice"

Hostname / IP address / Range*
10.102.70.43 REACHABILITY TEST INGEST TEST

Specify the host(s) to register.

Description

Description of the device.

Authentication ▼

Tagging Profiles ▼

outbound SSH ▲

- Disable
- Reset

CANCEL SAVE SAVE & DEPLOY

Configure TCP Port for Outbound SSH in Ingest

When you configure outbound SSH connections for device groups, you must configure TCP ports for each device group. To avoid opening multiple TCP ports, Paragon Insights 4.1.0 Release supports configuration of a single TCP port as ingest configuration for all outbound SSH connections across device groups that use iAgent (NETCONF).

To configure iAgent (NETCONF) TCP port in ingest:

1. Go to **Settings > Ingest**.
2. Select the **SSH** tab on the Ingest Settings page.
3. In the OutboundSSH page, enter the TCP port number.

You can use the toggle button to enable or disable the **Port** field.

4. Do one of the following:
 - Click **Save and Deploy** to save and commit the configuration in your existing network configurations.
 - Click **Save** to only save the configuration and not deploy it with the existing network.

NOTE: If you configure outbound SSH port for iAgent in device groups, that configuration takes precedence over the ingest configuration.

Connect a Device in Multiple Device Groups via Outbound SSH

You can connect a device that is managed in different device groups through outbound SSH by configuring multiple clients, where each client has the same port. In this case, you must create as many copies of the device as there are device groups. Each device must have the same port number.

As an example, consider device r0 (10.1.1.1) configured for device groups *dg1* and *dg2*. To connect 10.1.1.1 to both device groups via the same outbound SSH port, you can create one more device r1 (10.1.1.1) with the same IP and deploy it in dg2.

You must configure Paragon Insights for these ports in the respective device-groups. [Figure 16 on page 141](#) is an example device group configuration.

Figure 16: Edit Device Group Configuration

EDIT DEVICE GROUP ✕

Description ⓘ

Devices* ⓘ

▼ **Advanced**

Flow Ingest Deploy Nodes ⓘ

Reports ⓘ

Ingest Frequency Profiles ⓘ

Retention Policy ⓘ

> **Logging**

> **Notifications**

▼ **Ports**

Native Ports ⓘ ✓

Flow Ports ⓘ

sFlow Ports ⓘ

Syslog Ports ⓘ

SNMP Notification Ports ⓘ

Outbound SSH Ports ⓘ ✓

> **Summarization**

> **Data Rollup Summarization**

[Cancel](#) [Save](#) [Save and Deploy](#)

Using the following sample client configurations, device 10.1.1.1 can connect to two device groups using two outbound SSH clients with the same port.

```
set system services outbound-ssh client outbound-ssh1 device-id r0

set system services outbound-ssh client outbound-ssh1 10.1.1.1 port 2020

set system services outbound-ssh client outbound-ssh2 device-id r1

set system services outbound-ssh client outbound-ssh2 10.1.1.1 port 2020
```

NOTE: The 10.1.1.1 in the example denotes Paragon Insights (host) IP address.

iAgent - vCenter/ESXi Server Monitor

Since release 2.0.0, Paragon Insights has supported user-defined functions (UDFs) within fields. A user-defined function relies on python tables and views, and YAML configuration files to allow a user to define their own functions for processing telemetry data from Junos OS devices.

Starting with Release 3.2.0, Paragon Insights can use UDFs to process streamed data from VMWare's vCenter and ESXi server products.

We implement this feature by making use of VMWare's PyVmomi API and persistent SaltStack connections to that API. We use UDFs to process the vCenter/ESXi server data because:

- these servers do not respond to RPC with XML formatted text.
- it is extremely difficult to construct a table and view from the response data that these servers provide

SNMP

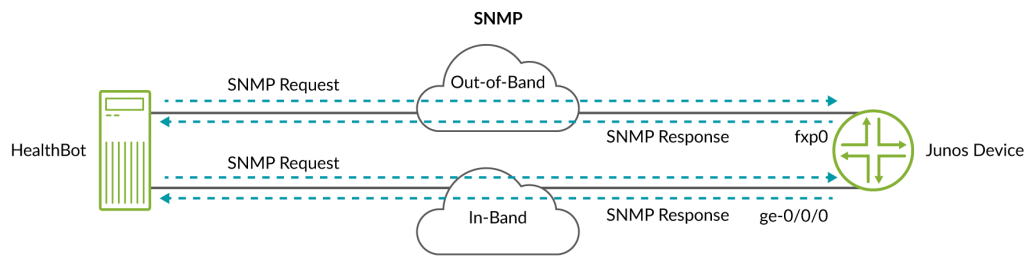
IN THIS SECTION

- [SNMP in Paragon Insights | 143](#)
- [Example: Creating a Rule using SNMP Ingest | 157](#)

- CONFIGURE NETWORK DEVICES | 158
- CREATE RULE, APPLY PLAYBOOK | 158
- Monitor the Devices | 171

SNMP is a widely known and accepted network management protocol that many network device manufacturers, including Juniper Networks, provide for use with their devices. It is a polling type protocol where network devices that are properly configured make configuration, diagnostic, and event information available to collectors, which must also be properly configured and authenticated. The collectors poll devices by sending specifically structured requests, called get requests, to retrieve data.

Paragon Insights supports SNMP as a sensor type, using standard get requests to gather statistics from the device. Paragon Insights makes requests over any available interface, whether in-band or out-of-band, and the device responds (when configured) with the requested data.



For information about SNMP as used on Junos OS devices, see [Understanding SNMP Implementation in Junos OS](#).

The sections below delve deeper into SNMP ingest configuration and all of the steps needed for Paragon Insights to successfully ingest SNMP data from a device or devices in a device group.

SNMP in Paragon Insights

IN THIS SECTION

- Configure a Rule Using SNMP Scalar Fields | 145
- SNMP Ingest Configurations | 146

Paragon Insights supports three methods of collecting telemetry data using SNMP. The ingest, also known as request-response, is a pull mode method in which Paragon Insights requests for telemetry

data from the devices. The trap and inform notifications are push mode methods in which the devices notify Paragon Insights about key performance indicator events that prevents the devices from functioning as expected.

Paragon Insights Release 4.0.0 supports SNMPv3 alongside the current SNMPv2c as an ingest method. Users with sp-admin role can select any version of SNMP in the Paragon Insights GUI.

SNMPv3 ingest provides you with an option to set authentication and privacy credentials to leverage the following features:

- Authentication – Identifies and verifies the origin of an SNMPv3 message.
- Privacy – Prevents packet analyzers from snooping the content of messages by encrypting them.
- Integrity – Ensures that the content of SNMP messages is not altered while in transit without authorization.

[Table 14 on page 144](#) lists the supported authentication and privacy algorithms in SNMPv3 protocol.

Table 14: Authentication and Privacy Algorithms

Feature	Algorithm
Supported authentication algorithms	MD5
	SHA
	SHA224
	SHA256
	SHA384
	SHA512
Supported privacy algorithms	DES
	AES
	AES192
	AES256
	AES192C
	AES256C

Paragon Insights also supports scalar object instances along with tabular objects in SNMP.

- The SNMP object can be scalar, tabular, or a combination of both in rules.

When you create a rule using SNMP ingest, you can add:

- Only scalar fields.
- A combination of tabular and scalar fields.
- A tabular column along with the index queried as a scalar object.

A tabular column queried as a scalar comes with the limitation that the index number does not refer to the same Object across all the devices when you configure the tabular field in rule.

For example, *IF-MIB::ifAdminStatus.16*. The ifAdminStatus is a column in **IF-MIB** table and *IF-MIB::ifAdminStatus.16* refers to the table column with index 16.

- Only tabular fields.
- A scalar object is identified by its MIB name (for example, **JUNIPER-MIB::scalarObjectName**) or as an OID.
- Paragon Insights validates a given scalar by checking the *MAX-ACCESS* property in the MIB definition.

If you find *MAX-ACCESS* in the MIB definition set to read-only, read-create, or read-write, then that object can be queried as a scalar.

When you query a table column with index as a scalar, you must enter the complete path to the scalar object as **MIB-Name::table column name:index number**.

For example:

```
IF-MIB::ifInOctets:16
```

NOTE: You can enter complete paths for more than one scalar object in the same query.

Configure a Rule Using SNMP Scalar Fields

The following example configuration of a rule has an SNMP tabular column as a scalar field and a second field with a scalar not indexed in the SNMP table.

To configure a rule with scalar fields:

1. Go to **Configuration > Rules**.

2. Click **+Add Rule** on top of the Rules page.

Fill in the topic and rule name, description, and synopsis.

By default, the new rule is saved in topic **external** unless you specify a topic name here.

3. Click **+Add Sensor** and enter a name for the sensor.
4. Select **SNMP** as Sensor Type and *30s* as Frequency.
5. Click **Add Scalar** and enter **IF-MIB::ifNumber** in the scalar field.

You can also enter the OID of the scalar object.

6. Click **Add Scalar** again and enter **IF-MIB::ifAdminStatus.16** in the scalar field.

NOTE: The index number is different from one device to another and from one system to another.

Before you configure a rule for a device, get the index number of the scalar object in that device.

For more information, see [SNMP index number](#).

7. Do one of the following:

- **Save** – Save your configuration changes but do not deploy the updated configuration. You can use this option when, for example, you are making several changes and want to deploy all your updates at the same time later.
- **Save & Deploy** – Save the rule configuration in the GUI and deploy the configuration on your production environment. The ingest starts collecting telemetry data based on the configuration changes.

SNMP Ingest Configurations

The SNMPv3 ingest can be set at the device or device group level, with device configuration taking precedence if the ingest is configured at both levels. The configuration of SNMPv3 and SNMPv2c is mutually exclusive.

NOTE: If a device is not configured for SNMP ingest, Paragon Insights uses SNMP v2c with **SNMP Community** set to *public* as the default settings.

NOTE: In Paragon Insights, the SNMPv2c and SNMPv3 ingest and trap configurations share the same workflow.

To configure SNMP ingest at the device level:

1. Click the **Configuration > Device** option in the left navigation bar.
2. Click the add device button (+).
3. Enter the necessary values in the text boxes and select the appropriate options for the device.

The following table describes the attributes in the **Add a Device** window:

Attributes	Description
Name	Name of the device. Default is hostname. (Required)
Hostname / IP Address / Range	Hostname or IP address of a single device. If you are providing a range of IP addresses, enter the IP address for the device that marks the start and end of the address range. (Required)
System ID to use for JTI	<p>Unique system identifier required for JTI native sensors. Junos devices use the following format: <code><host_name>:<jti_ip_address></code></p> <p>When a device has dual routing engines (REs), it might send different system IDs depending on which RE is primary. You can use a regular expression to match both system IDs.</p>
Flow Source IPs	Enter the IP address(es) that this device uses to send NetFlow data.
OpenConfig Port Number	Port number required for JTI OpenConfig sensors. The default value is 32767.
iAgent Port Number	Port number required for iAgent. The default value is 830.

(Continued)

Attributes	Description
Vendor	<p>Lists the vendor or supplier of the device you are using.</p> <p>The operating system you can select from the OS drop-down list depends on the vendor you select from the Vendor drop-down list.</p> <p>The following are the list of options you can choose from.</p> <ul style="list-style-type: none"> • Select Juniper from the vendor drop-down list to select either Junos or Junos Evolved operating systems from the OS drop-down list. • Select CISCO from the Vendor drop-down list to select either IOSXR or NXOS operating systems from the OS drop-down list. <p>With Paragon Insights Release 4.0.0, NXOS OS is also supported.</p> <p>NOTE: If you plan to use Cisco IOS XR devices, you must first configure the telemetry. For more information, see <i>Paragon Insights Installation Requirements</i></p> <ul style="list-style-type: none"> • Select Arista from the Vendor drop-down list to select EOS operating system from the OS drop-down list. • Select Paloalto from the Vendor drop-down list to select PANOS operating system from the OS drop-down list. • Select Linux from the Vendor drop-down list and you can enter the name of the operating system in the OS field. • If you select Other Vendor, the Vendor Name and OS Name fields are enabled. You can enter the name of the vendor of your choice in the Vendor

(Continued)

Attributes	Description
	<p>Name field and the corresponding operating system for the vendor in the OS field.</p> <p>Consider the following example. If the operating system of a vendor (listed in the Vendor drop-down list) is not listed (in the OS drop-down list), you can select the Other Vendor option to enter name of the vendor and operating system of your choice.</p> <p>Starting with Release 4.0.0, Paragon Insights supports Arista Networks, Paloalto Networks, and Linux vendors.</p>
Timezone	Timezone for this device, specified as <i>+</i> or <i>-</i> hh:mm. For example, +07:00
Syslog Source IPs	List of IP addresses for the device sending syslog messages to Paragon Insights. For example, 10.10.10.23, 192.168.10.100.
Syslog Hostnames	List of hostnames for the device sending syslog messages to Paragon Insights. For example, router1.example.com.
SNMP	
SNMP Port Number	Port number required for SNMP ingest (request-response) messages. The port number is set to the standard value of 161 .
SNMP Version	Select either v2c or v3 in the drop-down menu.

(Continued)

Attributes	Description
SNMP Community	<p>Enter an SNMP Community string for SNMPv2c ingest.</p> <p>In SNMPv2c, Community string is used to verify the authenticity of the ingest (request-response) message issued by the SNMP agent (devices such as routers, switches, servers, and so on).</p>
SNMPv3 Username	Enter a username for SNMPv3 ingest (request-response).
Authentication None	<p>This field appears if you selected v3 in SNMP Version field.</p> <p>Enable this option if you want to set SNMPv3 authentication to <i>None</i>.</p>
Privacy None	<p>This field appears if you selected v3 in SNMP Version field.</p> <p>Enable this option if you want to set SNMPv3 privacy protocol to <i>None</i>.</p>
SNMPv3 Authentication Protocol	<p>This field appears if you selected v3 in SNMP Version field and disabled <i>Authentication None</i>.</p> <p>Select an authentication protocol from the drop-down menu.</p> <p>SNMP authentication protocol hashes the SNMP <i>username</i> with the passphrase you enter. The hashed output is sent along with the ingest message. Insights again hashes the <i>username</i> with the passphrase you entered for authentication. If the output matches, the ingest message is further processed.</p>

(Continued)

Attributes	Description
SNMPv3 Authentication Passphrase	<p>This field appears if you selected v3 in SNMP Version field and disabled <i>Privacy None</i>.</p> <p>Enter a passphrase for SNMPv3 authentication.</p>
SNMPv3 Privacy Protocol	<p>Select a privacy protocol from the drop-down menu.</p> <p>Privacy algorithm encrypts the ingest message with the protocol passphrase so that the message cannot be read by an unauthorized application in the network.</p>
SNMPv3 Privacy Passphrase	<p>This field appears if you selected v3 in SNMP Version field and disabled <i>Privacy None</i>.</p> <p>Enter a passphrase to encrypt the ingest message.</p>

Authentication (Required either here or at Device Group level)

Password	<p>Username Authentication username.</p> <p>Password Authentication password.</p>
SSL	<p>Server Common Name Server name protected by the SSL certificate.</p> <p>CA Profile* Choose the applicable CA profile(s) from the drop-down list.</p> <p>Local Certificate* Choose the applicable local certificate profile(s) from the drop-down list.</p>

(Continued)

Attributes	Description
SSH	<p>SSH Key Profile* Choose the applicable SSH key profile(s) from the drop-down list.</p> <p>Username Authentication username.</p>

*To edit or view details about saved security profiles, go to the **Security** page under the **Settings** menu in the left navigation bar.

4. Click **Save** to save the configuration or click **Save and Deploy** to save and deploy the configuration. For information on how to use the Devices table, see *Monitor Device and Network Health*.

To configure SNMP ingest at the device-group level:

1. Click the **Configuration > Device Group** option in the left-nav bar.
2. Click the add group button (+).
3. Enter the necessary values in the text boxes and select the appropriate options for the device group.

The following table describes the attributes in the **Add a Device Group** window:

Attributes	Description
Name	Name of the device group. (Required)
Description	Description for the device group.
Devices	<p>Add devices to the device group from the drop-down list. (Required)</p> <p>Starting in Paragon Insights 4.0.0, you can add more than 50 devices per device group. However, the actual scale of the number of devices you can add depends on the available system resources.</p> <p>For example, consider that you want to create a device group of 120 devices. In releases earlier than release 4.0.0, it is recommended that you create three device groups of 50, 50, and 20 devices respectively. With Paragon Insights Release 4.0.0, you just create one device group.</p>

(Continued)

Attributes	Description
Native Ports	(Native GPB sensors only) List the port numbers on which the Junos Telemetry Interface (JTI) native protocol buffers connections are established.
Flow Ports	(NetFlow sensors only) List the port numbers on which the NetFlow data is received by Paragon Insights. The port numbers must be unique across the entire Paragon Insights installation.
Syslog Ports	Specify the UDP port(s) on which syslog messages are received by Paragon Insights.
Retention Policy	Select a retention policy from the drop-down list for time series data used by root cause analysis (RCA). By default, the retention policy is 7 days.
Reports	<p>In the Reports field, select one or more health report profile names from the drop-down list to generate reports for the device group. Reports include alarm statistics, device health data, as well as device-specific information (such as hardware and software specifications).</p> <p>To edit or view details about saved health report profiles, go to the System page under the Settings menu in the left-nav bar. The report profiles are listed under Report Settings.</p> <p>For more information, see <i>Alerts and Notifications</i>.</p>
SNMP	
SNMP Port Number	Port number required for SNMP notifications. The port number is set to the standard value of 161 .
SNMP Version	<p>Select either v2c or v3 in the drop-down menu.</p> <ul style="list-style-type: none"> • If you select v2c, the SNMP Community name field appears. The string used in v2c authentication is set to <i>public</i> by default. It is recommended that users change the community string. • If you select v3, you are given an option to set username, authentication and privacy methods, and authentication and privacy secrets.

(Continued)

Attributes	Description
SNMP Community	<p>Enter an SNMP Community string for SNMPv2c ingest.</p> <p>In SNMPv2c, Community string is used to verify the authenticity of the trap message issued by the SNMP agent.</p>
SNMPv3 Username	<p>Enter a username for SNMPv3 ingest messages.</p>
Authentication None	<p>This field appears if you selected v3 in SNMP Version field.</p> <p>Enable this option on if you want to set SNMPv3 authentication to <i>None</i>.</p>
Privacy None	<p>This field appears if you selected v3 in SNMP Version field.</p> <p>Enable this option on if you want to set SNMPv3 privacy protocol to <i>None</i>.</p>
SNMPv3 Authentication Protocol	<p>This field appears if you selected v3 in SNMP Version field and disabled <i>Authentication None</i>.</p> <p>Select an authentication protocol from the drop-down menu.</p> <p>SNMP authentication protocol hashes the SNMP <i>username</i> with the passphrase you enter. The hashed output is sent along with the trap notification message. Paragon Insights again hashes the <i>username</i> with the passphrase you entered for authentication. If the output matches, the trap notification is further processed.</p>
SNMPv3 Authentication Passphrase	<p>This field appears if you selected v3 in SNMP Version field and disabled <i>Privacy None</i>.</p> <p>Enter a passphrase to authenticate SNMPv3 ingest messages.</p>
SNMPv3 Privacy Protocol	<p>Select a privacy protocol from the drop-down menu.</p> <p>Privacy algorithm encrypts the ingest message with the protocol passphrase so that the message cannot be read by an unauthorized application in the network.</p>
SNMPv3 Privacy Passphrase	<p>This field appears if you selected v3 in SNMP Version field and disabled <i>Privacy None</i>.</p> <p>Enter a passphrase to encrypt the trap notification.</p>

(Continued)

Attributes	Description
Summarization	<p>To improve the performance and disk space utilization of the Paragon Insights time series database, you can configure data summarization methods to summarize the raw data collected by Paragon Insights. Use these fields to configure data summarization:</p> <p>Time Span The time span (in minutes) for which you want to group the data points for data summarization.</p> <p>Summarization Profiles Choose the data summarization profiles from the drop-down list for which you want to apply to the ingest data. To edit or view details about saved data summarization profiles, go to the Data Summarization Profiles page under the Settings menu in the left-nav bar.</p> <p>For more information, see <i>Configure Data Summarization</i>.</p>
Ingest Frequency	Select existing Ingest Frequency Profiles to override rule or sensor frequency settings.

Authentication(Required here or at Device level)

Password	<p>Username Authentication user name.</p> <p>Password Authentication password.</p>
SSL	<p>Server Common Name Server name protected by the SSL certificate.</p> <p>CA Profile* Choose the applicable CA profile(s) from the drop-down list.</p> <p>Local Certificate* Choose the applicable local certificate profile(s) from the drop-down list.</p>
SSH	<p>SSH Key Profile* Choose the applicable SSH key profile(s) from the drop-down list.</p> <p>Username Authentication username.</p>

(Continued)

Attributes	Description
Notifications	<ul style="list-style-type: none"> You can use the Alarm Manager feature to organize, track, and manage KPI event alarm notifications received from Paragon Insights devices. To receive Paragon Insights alarm notifications for KPI events that have occurred on your devices, you must first configure the notification delivery method for each KPI event severity level (Major, Minor, and Normal). Select the delivery method from the drop-down lists. <p>To edit or view details about saved delivery method profiles, go to the System page under the Settings menu in the left-nav bar. The delivery method profiles are listed under Notification Settings.</p> <p>For more information, see <i>Alerts and Notifications</i>.</p>
Logging Configuration	<p>You can collect different severity levels of logs for the running Paragon Insights services of a device group. Paragon Insights Release 4.0.0 supports log collection for SNMP notification.</p> <p>Use these fields to configure which log levels to collect:</p> <p>Global Log Level From the drop-down list, select the level of the log messages that you want to collect for every running Paragon Insights service for the device group. The level is set to error by default.</p> <p>Log Level for specific services Select the log level from the drop-down list for any specific service that you want to configure differently from the Global Log Level setting. The log level that you select for a specific service takes precedence over the Global Log Level setting.</p> <p>For more information, see <i>Logs for Paragon Insights Services</i>.</p>

(Continued)

Attributes	Description
Publish	<p>You can configure Paragon Insights to publish Paragon Insights sensor and field data for a specific device group:</p> <p>Destinations Select the publishing profiles that define the notification type requirements (such as authentication parameters) for publishing the data.</p> <p>To edit or view details about saved publishing profiles, go to the System page under the Settings menu in the left-nav bar. The publishing profiles are listed under Notification Settings.</p> <p>Field Select the Paragon Insights rule topic and rule name pairs that contain the field data you want to publish.</p> <p>Sensor (Device group only) Select the sensor paths or YAML tables that contain the sensor data you want to publish. No sensor data is published by default.</p>

*To edit or view details about saved security profiles, go to the Security page under the Settings menu in the left-nav bar.

4. Click **Save** to save the configuration or click **Save and Deploy** to save and deploy the configuration. For information on how to use the device group cards, see *Monitor Device and Network Health*.

Example: Creating a Rule using SNMP Ingest

To illustrate how to configure and use an SNMP sensor, consider a scenario where you want to:

- Monitor Routing Engine CPU, CPU average, and memory utilization for a device, using SNMP data
- Create a rule with triggers that indicate when utilization for any of the above elements goes above 80%

To implement this scenario, you will need to complete the following activities:

- ["CONFIGURE NETWORK DEVICES" on page 158](#)
- ["CREATE RULE, APPLY PLAYBOOK" on page 158](#)
- ["Monitor the Devices" on page 171](#)

The workflow is as follows:



CONFIGURE NETWORK DEVICES

NOTE: This example assumes you have already added your devices into Paragon Insights and assigned them to a device group.

If not already done, configure your network device(s) to accept SNMP ingest in Paragon Insights. See ["SNMP in Paragon Insights" on page 143](#) for steps to configure SNMP ingest.

CREATE RULE, APPLY PLAYBOOK

IN THIS SECTION

- [Configure a Rule Using an SNMP Sensor | 158](#)
- [Add the Rule to a Playbook | 169](#)
- [Apply the playbook to a device group | 170](#)

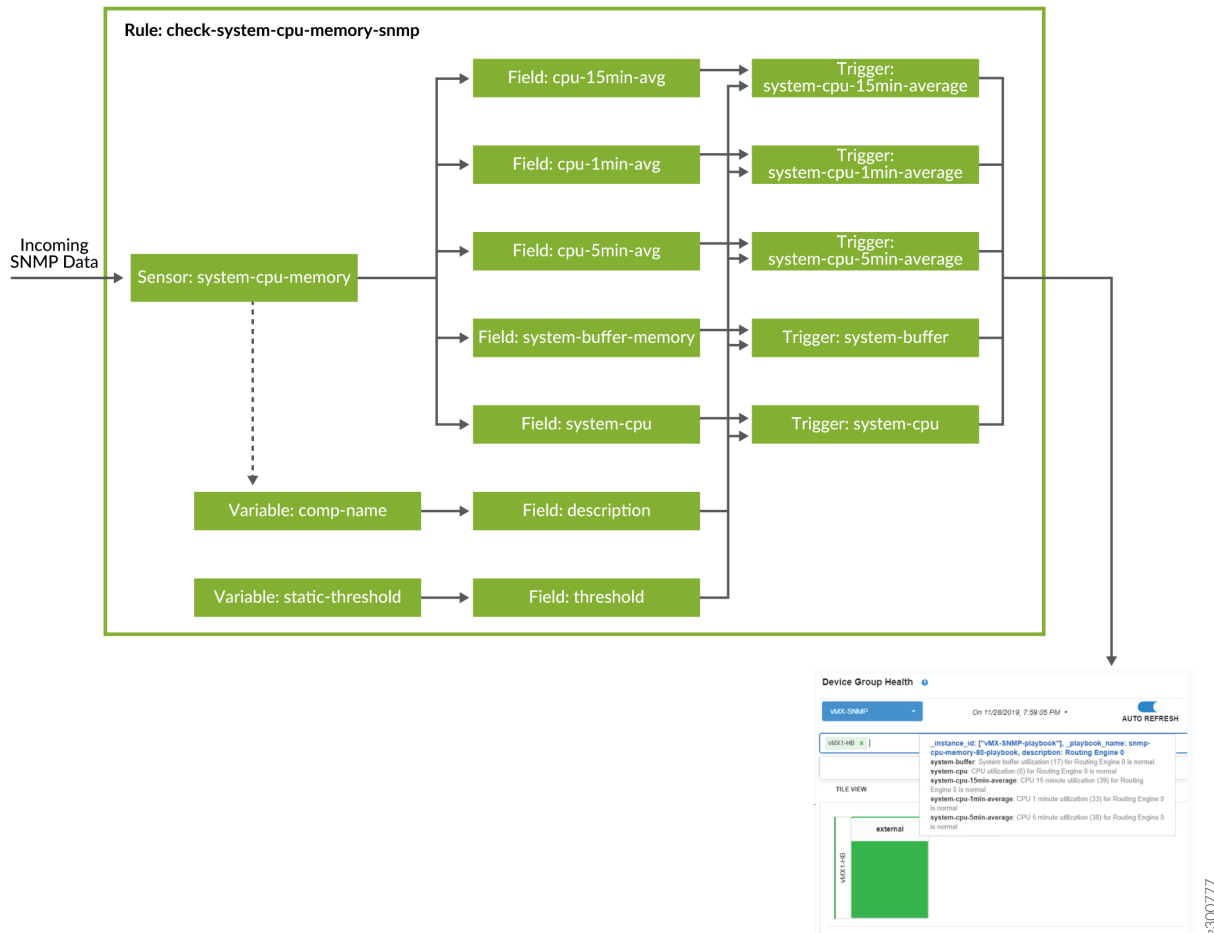
Configure a Rule Using an SNMP Sensor

You can now create a rule using SNMP as the sensor.

This rule includes multiple elements, as shown below:

- An SNMP sensor to ingest data
- Five fields extracting specific SNMP data of interest:
 - CPU utilization, memory utilization
 - CPU utilization averages - 1min, 5min, 15min
- A field representing a static value, used as a threshold
 - Value provided by a variable

- A field representing a description
 - Value provided by a variable; extracted from the SNMP messages
- Five triggers, indicating when CPU, CPU average, and memory utilization is higher than the threshold value



1. In the Paragon Insights GUI, click **Configuration > Rules** in the left-nav bar.
2. On the Rules page, click the **+ Add Rule** button.
3. On the page that appears, in the top row of the rule window, set the rule name. In this example, rule name is **check-system-cpu-memory-snmp**.
4. Add a description and synopsis if you wish.
5. Click the **+ Add sensor** button and enter the following parameters to configure the sensor, **system-cpu-memory**:
 - Name is user-defined
 - The sensor is using the Juniper SNMP MIB table **jnxOperatingTable**

- Paragon Insights polls the device group for table data every **60** seconds

The screenshot shows the configuration for a sensor named 'system-cpu-memory'. The sensor type is 'SNMP'. The table is 'JUNIPER-MIB::jnxOperatingTable' and the frequency is '60s'. A 'Delete system-cpu-memory' button is present in the top right corner.

6. Now move to the Variables tab, click the **+ Add variable** button and enter the following parameters to configure the first variable, **comp-name**:

- Matches any string that includes “Routing Engine”
- Referenced later in field **description**

The screenshot shows the configuration for a variable named 'comp-name'. The default value is '*.Routing Engine.*' and the type is 'String'. The description is 'Collects all routing engines CPU and memory utilization'. A 'Delete comp-name' button is present in the top right corner.

7. Click the **+ Add variable** button once more and enter the following parameters to configure the second variable, **static-threshold**:

- Represents a (default) static value of “80”; in this case, 80%
- Referenced later in field **threshold**

The screenshot shows the configuration for a variable named 'static-threshold'. The default value is '80' and the type is 'Integer'. The description is 'Static threshold value for CPU and memory utilization'. A 'Delete static-threshold' button is present in the top right corner.

8. Now move to the Fields tab, click the **+ Add field** button and enter the following parameters to configure the first field, **cpu-15min-avg**:

- Field names are user-defined
- Extracts **jnxOperating15MinLoadAvg** value from SNMP table configured in the sensor

- [jnxOperating15MinLoadAvg](#) - CPU Load Average (as a % value) over the last 15 minutes

Sensors **Fields** Vectors Variables Functions Triggers Rule Properties

[+ Add field](#) [Delete cpu-15min-avg](#)

cpu-15min-avg
cpu-1min-avg
cpu-5min-avg
description
system-buffer-memory
system-cpu
threshold

Field name* [?](#)
cpu-15min-avg

Description [?](#)
System cpu utilization 15 min average value

Field type
integer

Add to rule key [?](#)

Ingest type (Field source)
Sensor

Sensor **Path*** **Data if missing**

system-cpu-memory jnxOperating15MinLoadAvg Zero suppression Default value

9. Click the **+ Add field** button again and enter the following parameters to configure the second field, **cpu-1min-avg**:

- Extracts [jnxOperating1MinLoadAvg](#) value from SNMP table
- [jnxOperating1MinLoadAvg](#) - CPU Load Average (as a % value) over the last 1 minute

Sensors **Fields** Vectors Variables Functions Triggers Rule Properties

[+ Add field](#) [Delete cpu-1min-avg](#)

cpu-15min-avg
cpu-1min-avg
cpu-5min-avg
description
system-buffer-memory
system-cpu
threshold

Field name* [?](#)
cpu-1min-avg

Description [?](#)
System cpu utilization 1 min average value

Field type
integer

Add to rule key [?](#)

Ingest type (Field source)
Sensor

Sensor **Path*** **Data if missing**

system-cpu-memory jnxOperating1MinLoadAvg Zero suppression Default value

10. Click the **+ Add field** button again and enter the following parameters to configure the third field, **cpu-5min-avg**:

- Extracts [jnxOperating5MinLoadAvg](#) value from SNMP table
- [jnxOperating5MinLoadAvg](#) - CPU Load Average (as a % value) over the last 5 minutes

Sensors **Fields** Vectors Variables Functions Triggers Rule Properties

[+ Add field](#) [Delete cpu-5min-avg](#)

Field name* [?](#)

cpu-15min-avg
cpu-1min-avg
cpu-5min-avg
description
system-buffer-memory
system-cpu
threshold

cpu-5min-avg

Description [?](#)

System cpu utilization 5 min average value.

Field type

integer

Add to rule key [?](#)

Ingest type (Field source)

Sensor

Sensor **Path*** **Data if missing**

system-cpu-memory jnxOperating5MinLoadAvg Zero suppression Default value

11. Click the **+ Add field** button again and enter the following parameters to configure the fourth field, **description**:

- Extracts **jnxOperatingDescr** value from SNMP table
- **jnxOperatingDescr** - name or description; for example, "Routing Engine 0", "FPC 0", etc.
- The expression references the variable **comp-name**; filters the data further to retain only the values that include the string "Routing Engine"
- Matching values will act as keys; each key gets a colored block in device health view

Sensors **Fields** Vectors Variables Functions Triggers Rule Properties

+ Add field **Delete description**

Field name* ? description

Description ? Collect operating description name which contains Routing Engine in the jnxOperatingDescr

Field type string

Add to rule key ?

Ingest type (Field source) Sensor

Sensor system-cpu-memory **Path*** jnxOperatingDescr Zero suppression **Data if missing** Default value

Where (filter using expression)

Where	Operator	Value
jnxOperatingDescr	regex	{{comp-name}}/

+ Add Expression

12. Click the **+ Add field** button again and enter the following parameters to configure the fifth field, **system-buffer-memory**:

- Extracts **jnxOperatingBuffer** value from SNMP table
- **jnxOperatingBuffer** - buffer pool utilization (as a % value)

Sensors **Fields** Vectors Variables Functions Triggers Rule Properties

+ Add field **Delete system-buffer-memory**

Field name* ? system-buffer-memory

Description ? System memory buffer utilization value

Field type integer

Add to rule key ?

Ingest type (Field source) Sensor

Sensor system-cpu-memory **Path*** jnxOperatingBuffer Zero suppression **Data if missing** Default value

13. Click the **+ Add field** button again and enter the following parameters to configure the sixth field, **system-cpu**:

- Extracts **jnxOperatingCPU** value from SNMP table
- **jnxOperatingCPU** - CPU utilization (as a % value)

Sensors **Fields** Vectors Variables Functions Triggers Rule Properties

+ Add field **Delete system-cpu**

Field name* ?
system-cpu

Description ?
System CPU utilization value

Field type
integer

Add to rule key ?

Ingest type (Field source)
Sensor

Sensor Path* ? Zero suppression Data if missing
system-cpu-memory jnxOperatingCPU Default value

14. Click the **+ Add field** button once more and enter the following parameters to configure the seventh field, **threshold**:

- The expression references the variable **static-threshold**, giving this field the (default) integer value "80"
- Referenced later in triggers

Sensors **Fields** Vectors Variables Functions Triggers Rule Properties

+ Add field **Delete threshold**

Field name* ?
threshold

Description ?
System CPU utilization threshold value

Field type
integer

Add to rule key ?

Ingest type (Field source)
Constant

Constant value* ?
{{static-threshold}}

15. Now move to the Triggers tab, click the **+ Add trigger** button and enter the following parameters to configure the first trigger, **system-buffer**:


- Trigger names are user-defined


- Trigger logic runs every **90** seconds
- Evaluate terms in sequence; when a term's conditions are met, show its color and message on the device health pages
- When system memory buffer utilization (the value in field **system-buffer-memory**) is greater than 80 (the value in field **threshold**), set color to red and show related message
- Otherwise, set color to green and show related message


Sensors Fields Vectors Variables Functions **Triggers** Rule Properties

+ Add trigger **Delete system-buffer**



system-buffer
system-cpu
system-cpu-15min-average
system-cpu-1min-average
system-cpu-5min-average

Trigger name  system-buffer

Frequency  90s


Term **is-system-buffer-memory-utilization-ab** 

WHEN

Left operand	Operator	Right operand	All  in time range
<input type="text" value="\$system-buffer-memory"/>	<input type="text" value=">"/>	<input type="text" value="\$threshold"/>	<input type="text" value="Enter a time range"/> 


+ Add Condition


THEN

Color 

Message
System buffer utilization(\$system-buffer-memory) for \$description has crossed static threshold value (\$threshold)

Evaluate next term


 Functions can be used as Trigger actions too, define them using the 'Functions' menu at the top.

Term **is-system-buffer-memory-utilization-is-** 

WHEN

+ Add Condition

THEN

Color 

Message
System buffer utilization (\$system-buffer-memory) for \$description is normal

Evaluate next term


- 16.** Click the **+ Add trigger** button again and enter the following parameters to configure the second trigger, **system-cpu**:


- Trigger logic runs every **90** seconds
- When CPU utilization (the value in field **system-cpu**) is greater than 80 (the value in field **threshold**), set color to red and show related message
- Otherwise, set color to green and show related message


Sensors Fields Vectors Variables Functions **Triggers** Rule Properties

+ Add trigger



system-buffer
system-cpu
system-cpu-15min-average
system-cpu-1min-average
system-cpu-5min-average

Trigger name  system-cpu **Delete system-cpu**

Frequency  90s


Term is-system-cpu-utilization-abnormal 

WHEN

Left operand	Operator	Right operand	All  in time range
\$system-cpu	>	\$threshold	Enter a time range 


+ Add Condition


THEN

Color 

Message
CPU utilization (\$system-cpu) for \$description has crossed static threshold value (\$threshold)

Evaluate next term


 Functions can be used as Trigger actions too, define them using the 'Functions' menu at the top.

Term is-system-cpu-utilization-normal 

WHEN

+ Add Condition

THEN

Color 

Message
CPU utilization (\$system-cpu) for \$description is normal

Evaluate next term

17. Click the click the **+ Add trigger** button again and enter the following parameters to configure the third trigger, **system-cpu-15min-average**:

- Trigger logic runs every **90** seconds

- When CPU 15min utilization average (the value in field **cpu-15min-avg**) is greater than or equal to 80 (the value in field **threshold**), set color to red and show related message
- Otherwise, set color to green and show related message

The screenshot displays the 'Triggers' configuration page. On the left, a sidebar lists available triggers: system-buffer, system-cpu, system-cpu-15min-average (selected), system-cpu-1min-average, and system-cpu-5min-average. The main area shows the configuration for the selected trigger, 'system-cpu-15min-average'. The 'Trigger name' is 'system-cpu-15min-average'. The 'Frequency' is '90s'. The 'Term' is 'is-cpu-15m-average-util-abnormal'. The 'WHEN' section contains a condition: Left operand '\$cpu-15min-avg', Operator '>=', Right operand '\$threshold', and 'All' in time range. The 'THEN' section has 'Color' set to red and 'Message' set to 'CPU 15 minute utilization (\$cpu-15min-avg) for \$description has crossed static threshold value (\$threshold)'. Below this, there is an 'Evaluate next term' toggle and an information box stating 'Functions can be used as Trigger actions too, define them using the 'Functions' menu at the top.' A second trigger configuration is partially visible below, for 'system-cpu-15m-average-util-normal', with a green color and a normal message.

18. Click the click the **+ Add trigger** button again and enter the following parameters to configure the fourth trigger, **system-cpu-1min-average**:
- Trigger logic runs every **90** seconds
 - When CPU 1min utilization average (the value in field **cpu-1min-avg**) is greater than or equal to 80 (the value in field **threshold**), set color to red and show related message

- Otherwise, set color to green and show related message

Sensors Fields Vectors Variables Functions **Triggers** Rule Properties

+ Add trigger Delete system-cpu-1min-average

system-buffer
system-cpu
system-cpu-15min-average
system-cpu-1min-average
system-cpu-5min-average

Trigger name ?
system-cpu-1min-average

Frequency ?
90s

Term is-cpu-1m-average-util-abnormal ×

WHEN

Left operand	Operator	Right operand	All ▼ in time range
\$cpu-1min-avg	>=	\$threshold	Enter a time range ⊖

+ Add Condition

THEN

Color
■

Message
CPU 1 minute utilization (\$cpu-1min-avg) for \$description has crossed static threshold value (\$threshold)

Evaluate next term

i Functions can be used as Trigger actions too, define them using the 'Functions' menu at the top.

Term is-cpu-1m-average-util-normal ×

WHEN

+ Add Condition

THEN

Color
■

Message
CPU 1 minute utilization (\$cpu-1min-avg) for \$description is normal


Evaluate next term


19. Click the click the **+ Add trigger** button once more and enter the following parameters to configure the fifth trigger, called **system-cpu-5min-average**:
- Trigger logic runs every **90** seconds
 - When CPU 5min utilization average (the value in field **cpu-5min-avg**) is greater than or equal to **80** (the value in field **threshold**), set color to red and show related message
 - Otherwise, set color to green and show related message


Sensors Fields Vectors Variables Functions **Triggers** Rule Properties

+ Add trigger **Delete system-cpu-5min-average**






system-buffer
system-cpu
system-cpu-15min-average
system-cpu-1min-average
system-cpu-5min-average

Trigger name  system-cpu-5min-average

Frequency  90s


Term is-cpu-5m-average-util-abnormal 

WHEN

Left operand	Operator	Right operand	All  in time range
\$cpu-5min-avg 	>= 	\$threshold 	Enter a time range 


+ Add Condition


THEN

Color 

Message
CPU 5 minute utilization (\$cpu-5min-avg) for \$description has crossed static threshold value (\$threshold)

Evaluate next term


 Functions can be used as Trigger actions too, define them using the 'Functions' menu at the top.

Term is-cpu-5m-average-util-normal 

WHEN

+ Add Condition

THEN

Color 

Message
CPU 5 minute utilization (\$cpu-5min-avg) for \$description is normal

Evaluate next term

20. At the upper right of the window, click the **+ Save & Deploy** button.

Add the Rule to a Playbook

With the rule created, you can now add it to a playbook. For this example, create a new playbook to hold the new rule.

1. Click **Configuration > Playbooks** in the left-navigation bar.
2. On the Playbooks page, click the **+ Create Playbook** button.
3. On the page that appears, enter the following parameters:

Create Playbook

Name* ⓘ

Synopsis ⓘ

Detailed description about the playbook and usage details

Rules* ⓘ

Cancel Save Save & Deploy

4. Click **Save & Deploy**.

Apply the playbook to a device group

To make use of the playbook, apply it to a device group.

1. On the Playbooks page, click the **Apply (Airplane)** icon for the playbook you configured above.
2. On the page that appears:
 - Enter a playbook instance name
 - Select the desired device group
 - (Optional) If desired, you can adjust the variables for this playbook instance to use different values than the defaults configured in the rule
 - Click **Run Instance**

Run Playbook: snmp-cpu-memory-80-playbook

Name of Playbook instance *

Run on schedule

Rules

Device Group

vMX-SNMP

Devices

vMX1-HB
MX240

Variable values for Group vMX-SNMP

external/check-snmp-system-cpu-memory

Comp Name

Collects all routing engines CPU and memory utilization

Static Threshold

Static threshold value for CPU and memory utilization

Cancel Save Instance Run Instance

3. On the Playbooks page, confirm that the playbook instance is running. Note that the playbook instance may take some time to activate.

Monitor the Devices

With the playbook applied, you can begin to monitor the devices.

1. Click **Monitor > Device Group Health** in the left-nav bar.
2. Select the device group to which you applied the playbook from the **Device Group** pull-down menu.
3. Select one or more of the devices to monitor.
4. In the Tile View, hover your mouse over one of the **external** tiles.
 - **external** is the topic name under which the rule was created
 - Each colored block represents a key and its related values
 - The mouse-over window shows information related to the given key, with the triggers listed inside



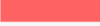

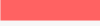




5. In the Table View, try out the various filters and sorting options.

- Each trigger is listed as a KPI

TABLE VIEW

Time
 Device
 Group
 Topic
 Keys
 KPI
 Status
 Message

Topic	Keys	KPI	Status	Message
external			ALL ▾	
external	_instance_i...	system-buffer		System buffer utilization (11) for Routing Engine 1 is no...
external	_instance_i...	system-cpu		CPU utilization (25) for Routing Engine 1 is normal
external	_instance_i...	system-cpu...		CPU 15 minute utilization (111) for Routing Engine 1 h...
external	_instance_i...	system-cpu...		CPU 1 minute utilization (127) for Routing Engine 1 ha...
external	_instance_i...	system-cpu...		CPU 5 minute utilization (118) for Routing Engine 1 ha...
external	_instance_i...	system-buffer		System buffer utilization (17) for Routing Engine 0 is n...
external	_instance_i...	system-cpu		CPU utilization (20) for Routing Engine 0 is normal

Release History Table

Release	Description
4.1.0	Paragon Insights 4.1.0 Release supports configuration of a single TCP port as ingest configuration for all outbound SSH connections across device groups that use iAgent (NETCONF).
4.0.0	Paragon Insights Release 4.0.0 supports SNMPv3 alongside the current SNMPv2c as an ingest method.
4.0.0	Starting in Paragon Insights 4.0.0, you can add more than 50 devices per device group.
4.0.0	Paragon Insights Release 4.0.0 supports log collection for SNMP notification.
3.2.0	Starting with Release 3.2.0, HealthBot can use iAgent connections that are device-initiated using outbound SSH
3.2.0	Starting with Release 3.2.0, Paragon Insights can use UDFs to process streamed data from VMWare's vCenter and ESXi server products
3.1.0	Starting in HealthBot Release 3.1.0, iAgent functionality is extended to third party devices.
2.0.0	Since release 2.0.0, Paragon Insights has supported user-defined functions (UDFs) within fields.