

HealthBot User Guide

Published
2021-02-17

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.

HealthBot User Guide

Copyright © 2021 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 the Documentation | ix

Documentation and Release Notes | ix

Documentation Conventions | ix

Documentation Feedback | xii

Requesting Technical Support | xii

Self-Help Online Tools and Resources | xiii

Creating a Service Request with JTAC | xiii

1

Introduction to HealthBot

HealthBot Overview | 15

Benefits of HealthBot | 15

Closed-Loop Automation | 16

Main Components of HealthBot | 17

HealthBot Health Monitoring | 17

HealthBot Root Cause Analysis | 18

HealthBot Log File Analysis | 19

HealthBot Concepts | 20

HealthBot Data Collection Methods | 21

Data Collection - 'Push' Model | 21

Data Collection - 'Pull' Model | 22

HealthBot Topics | 22

HealthBot Rules - Basics | 23

HealthBot Rules - Deep Dive | 25

Rules | 25

Sensors | 28

Fields | 28

Vectors | 30

Variables | 30

Functions | 31

Triggers | 31

Tagging | 34

Rule Properties | 35

HealthBot Playbooks | 35

Healthbot Tagging | 36

Overview | 36

HealthBot Tagging Terminology | 37

How It Works | 41

Examples | 42

HealthBot Time Series Database (TSDB) | 48

Historical Context | 48

TSDB Improvements | 49

Database Sharding | 50

Database Replication | 51

Database Reads and Writes | 52

Manage TSDB Options in the HealthBot GUI | 53

HealthBot CLI Configuration Options | 55

HealthBot Machine Learning (ML) | 56

HealthBot Machine Learning Overview | 56

Understanding HealthBot Anomaly Detection | 57

Field | 57

Algorithm | 58

Learning period | 58

Pattern periodicity | 59

Understanding HealthBot Outlier Detection | 59

Dataset | 60

Algorithm | 60

Sigma coefficient (k-fold-3sigma only) | 62

Sensitivity | 62

Learning period | 62

Understanding HealthBot Predict | 62

Field | 63

Algorithm | 63

Learning period | 63

Pattern periodicity | 63

Prediction offset | 63

HealthBot Rule Examples | 64

HealthBot Anomaly Detection Example | 64

HealthBot Outlier Detection Example | 73

Frequency Profiles and Offset Time | 78

Frequency Profiles | 78

Configuration Using HealthBot GUI | 79

Configuration Using HealthBot CLI | 81

Apply a Frequency Profile Using the HealthBot GUI | 81

Apply a Frequency Profile Using the HealthBot CLI | 83

Offset Time Unit | 83

Offset Used in Formulas | 84

Offset Used in References | 85

Offset Used in Vectors | 86

Offset Used in Triggers | 88

Offset Used in Trigger Reference | 89

HealthBot Licensing | 91

HealthBot Licensing Overview | 91

Managing HealthBot Licenses | 94

Add a License to HealthBot | 94

View Licensing Status in HealthBot | 94

2

Management and Monitoring

Manage HealthBot Users and Groups | 98

User Management | 98

Group Management | 99

Limitations | 104

Manage Devices, Device Groups, and Network Groups | 104

Adding a Device | 106

Editing a Device | 109

Adding a Device Group | 109

Editing a Device Group | 113

Configuring a Retention Policy for the Time Series Database | 113

Adding a Network Group | 114

Editing a Network Group | 117

HealthBot Rules and Playbooks | 118

Add a Pre-Defined Rule | 119

Create a New Rule Using the HealthBot GUI | 119

Rule Filtering | 121

Sensors | 123

Fields | 125

Vectors | 128

Variables | 130

Functions | 131

Triggers | 133

Rule Properties | 136

Edit a Rule | 136

Add a Pre-Defined Playbook | 137

Create a New Playbook Using the HealthBot GUI | 138

Edit a Playbook | 139

Manage Playbook Instances | 140

View Information About Playbook Instances | 141

Create a Playbook Instance | 143

Manually Pause or Play a Playbook Instance | 145

Create a Schedule to Automatically Play/Pause a Playbook Instance | 146

Monitor Device and Network Health | 148

Dashboard | 149

Health | 152

Network Health | 165

Graph Page | 165

Alarms and Notifications | 181

Generate Alarm Notifications | 181

Manage Alarms Using Alarm Manager | 190

Stream Sensor and Field Data from HealthBot | 195

Generate Reports | 200

Configure a Secure Data Connection for HealthBot Devices | 216

Configure Security Profiles for SSL and SSH Authentication | 217

Configure Security Authentication for a Specific Device or Device Group | 218

Configure Data Summarization | 219

Creating a Data Summarization Profile | 220

Applying Data Summarization Profiles to a Device Group | 221

Modify the UDA and UDF Engines | 222

Overview | 222

How it Works | 223

Usage Notes | 224

Configuration | 225

SIMULATE | 225

MODIFY | 226

ROLLBACK | 227

Logs for HealthBot Services | 227

Configure Service Log Levels for a Device Group or Network Group | 228

Download Logs for HealthBot Services | 229

Troubleshooting | 230

HealthBot Self Test | 230

Overview | 230

Other Uses for the Self Test Tool | 231

Usage Notes | 231

How to Use the Self Test Tool | 232

Device Reachability Test | 232

Overview | 232

Usage Notes | 233

How to Use the Device Reachability Tool | 233

Ingest Connectivity Test | 234**Overview | 234****Usage Notes | 235****How to Use the Ingest Connectivity Tool | 235****Debug No-Data | 236****Overview | 236****Usage Notes | 237****How to Use the Debug No-Data Tool | 238****HealthBot Configuration – Backup and Restore | 240****Back Up the Configuration | 240****Restore the Configuration | 240****Backup or Restore the Time Series Database (TSDB) | 241**

About the Documentation

IN THIS SECTION

- Documentation and Release Notes | ix
- Documentation Conventions | ix
- Documentation Feedback | xii
- Requesting Technical Support | xii

Use this guide to understand the features you can configure and the tasks you can perform from the HealthBot web UI.

Documentation and Release Notes

To obtain the most current version of all Juniper Networks[®] technical documentation, see the product documentation page on the Juniper Networks website at <https://www.juniper.net/documentation/>.

If the information in the latest release notes differs from the information in the documentation, follow the product Release Notes.

Juniper Networks Books publishes books by Juniper Networks engineers and subject matter experts. These books go beyond the technical documentation to explore the nuances of network architecture, deployment, and administration. The current list can be viewed at <https://www.juniper.net/books>.

Documentation Conventions

[Table 1 on page x](#) defines notice icons used in this guide.

Table 1: Notice Icons







Icon	Meaning	Description
	Informational note	Indicates important features or instructions.
	Caution	Indicates a situation that might result in loss of data or hardware damage.
	Warning	Alerts you to the risk of personal injury or death.
	Laser warning	Alerts you to the risk of personal injury from a laser.
	Tip	Indicates helpful information.
	Best practice	Alerts you to a recommended use or implementation.

Table 2 on page x defines the text and syntax conventions used in this guide.

Table 2: Text and Syntax Conventions

Convention	Description	Examples
Bold text like this	Represents text that you type.	To enter configuration mode, type the configure command: user@host> configure
Fixed-width text like this	Represents output that appears on the terminal screen.	user@host> show chassis alarms No alarms currently active
<i>Italic text like this</i>	<ul style="list-style-type: none"> Introduces or emphasizes important new terms. Identifies guide names. Identifies RFC and Internet draft titles. 	<ul style="list-style-type: none"> A policy <i>term</i> is a named structure that defines match conditions and actions. <i>Junos OS CLI User Guide</i> RFC 1997, <i>BGP Communities Attribute</i>

Table 2: Text and Syntax Conventions (*continued*)

Convention	Description	Examples
<i>Italic text like this</i>	Represents variables (options for which you substitute a value) in commands or configuration statements.	Configure the machine's domain name: [edit] root@# set system domain-name <i>domain-name</i>
Text like this	Represents names of configuration statements, commands, files, and directories; configuration hierarchy levels; or labels on routing platform components.	<ul style="list-style-type: none"> To configure a stub area, include the stub statement at the [edit protocols ospf area area-id] hierarchy level. The console port is labeled CONSOLE.
< > (angle brackets)	Encloses optional keywords or variables.	stub <default-metric <i>metric</i>>;
(pipe symbol)	Indicates a choice between the mutually exclusive keywords or variables on either side of the symbol. The set of choices is often enclosed in parentheses for clarity.	broadcast multicast (<i>string1</i> <i>string2</i> <i>string3</i>)
# (pound sign)	Indicates a comment specified on the same line as the configuration statement to which it applies.	rsvp { # Required for dynamic MPLS only
[] (square brackets)	Encloses a variable for which you can substitute one or more values.	community name members [<i>community-ids</i>]
Indentation and braces ({ })	Identifies a level in the configuration hierarchy.	[edit] routing-options { static { route default { nexthop <i>address</i> ; retain; } } }
;(semicolon)	Identifies a leaf statement at a configuration hierarchy level.	

GUI Conventions

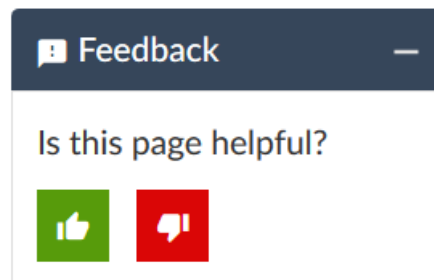
Table 2: Text and Syntax Conventions (*continued*)

Convention	Description	Examples
Bold text like this	Represents graphical user interface (GUI) items you click or select.	<ul style="list-style-type: none"> In the Logical Interfaces box, select All Interfaces. To cancel the configuration, click Cancel.
> (bold right angle bracket)	Separates levels in a hierarchy of menu selections.	In the configuration editor hierarchy, select Protocols>Ospf .

Documentation Feedback

We encourage you to provide feedback so that we can improve our documentation. You can use either of the following methods:

- Online feedback system—Click TechLibrary Feedback, on the lower right of any page on the [Juniper Networks TechLibrary](#) site, and do one of the following:



- Click the thumbs-up icon if the information on the page was helpful to you.
- Click the thumbs-down icon if the information on the page was not helpful to you or if you have suggestions for improvement, and use the pop-up form to provide feedback.
- E-mail—Send your comments to techpubs-comments@juniper.net. Include the document or topic name, URL or page number, and software version (if applicable).

Requesting Technical Support

Technical product support is available through the Juniper Networks Technical Assistance Center (JTAC). If you are a customer with an active Juniper Care or Partner Support Services support contract, or are

covered under warranty, and need post-sales technical support, you can access our tools and resources online or open a case with JTAC.

- JTAC policies—For a complete understanding of our JTAC procedures and policies, review the *JTAC User Guide* located at <https://www.juniper.net/us/en/local/pdf/resource-guides/7100059-en.pdf>.
- Product warranties—For product warranty information, visit <https://www.juniper.net/support/warranty/>.
- JTAC hours of operation—The JTAC centers have resources available 24 hours a day, 7 days a week, 365 days a year.

Self-Help Online Tools and Resources

For quick and easy problem resolution, Juniper Networks has designed an online self-service portal called the Customer Support Center (CSC) that provides you with the following features:

- Find CSC offerings: <https://www.juniper.net/customers/support/>
- Search for known bugs: <https://prsearch.juniper.net/>
- Find product documentation: <https://www.juniper.net/documentation/>
- Find solutions and answer questions using our Knowledge Base: <https://kb.juniper.net/>
- Download the latest versions of software and review release notes: <https://www.juniper.net/customers/csc/software/>
- Search technical bulletins for relevant hardware and software notifications: <https://kb.juniper.net/InfoCenter/>
- Join and participate in the Juniper Networks Community Forum: <https://www.juniper.net/company/communities/>
- Create a service request online: <https://myjuniper.juniper.net>

To verify service entitlement by product serial number, use our Serial Number Entitlement (SNE) Tool: <https://entitlementsearch.juniper.net/entitlementsearch/>

Creating a Service Request with JTAC

You can create a service request with JTAC on the Web or by telephone.

- Visit <https://myjuniper.juniper.net>.
- Call 1-888-314-JTAC (1-888-314-5822 toll-free in the USA, Canada, and Mexico).

For international or direct-dial options in countries without toll-free numbers, see <https://support.juniper.net/support/requesting-support/>.

1

CHAPTER

Introduction to HealthBot

HealthBot Overview | **15**

HealthBot Concepts | **20**

Healthbot Tagging | **36**

HealthBot Time Series Database (TSDB) | **48**

HealthBot Machine Learning (ML) | **56**

Frequency Profiles and Offset Time | **78**

HealthBot Licensing | **91**

HealthBot Overview

IN THIS SECTION

- [Benefits of HealthBot | 15](#)
- [Closed-Loop Automation | 16](#)
- [Main Components of HealthBot | 17](#)

HealthBot is a highly automated and programmable device-level diagnostics and network analytics tool that provides consistent and coherent operational intelligence across network deployments. Integrated with multiple data collection methods (such as Junos Telemetry Interface, NETCONF, syslog, and SNMP), HealthBot aggregates and correlates large volumes of time-sensitive telemetry data, providing a multidimensional and predictive view of the network. Additionally, HealthBot translates troubleshooting, maintenance, and real-time analytics into an intuitive user experience to give network operators actionable insights into the health of an individual device and the overall network.

Benefits of HealthBot

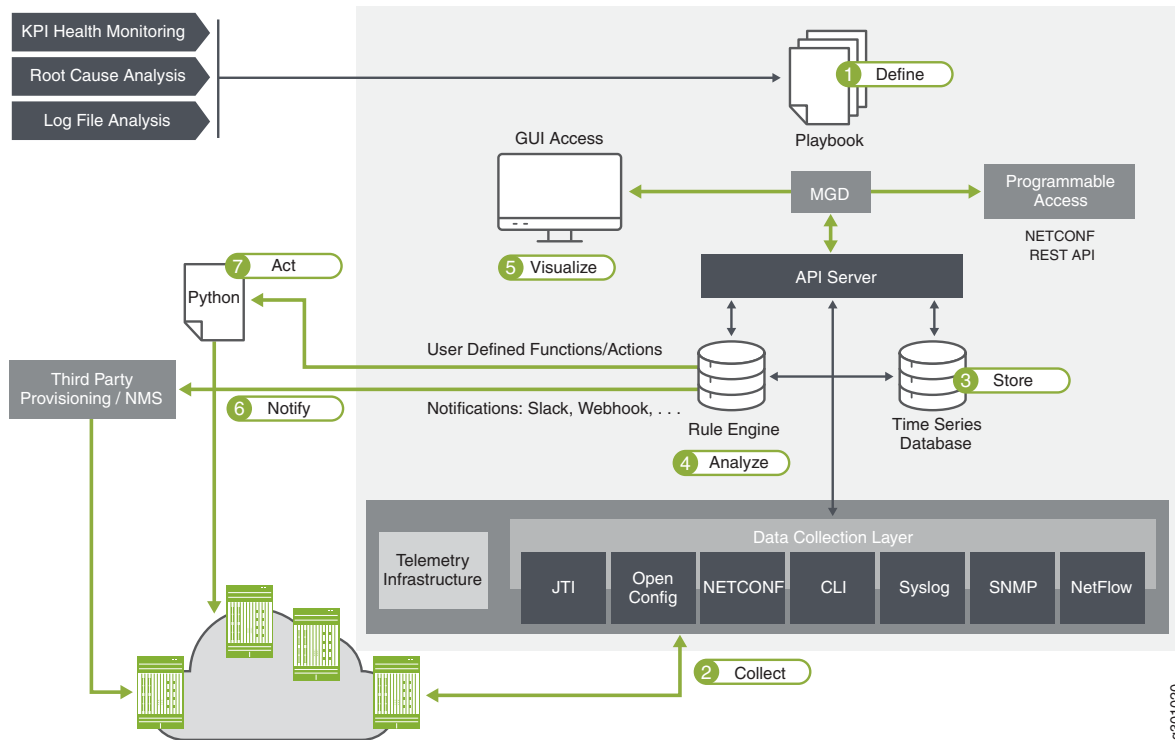
- **Customization**—Provides a framework to define and customize health profiles, allowing truly actionable insights for the specific device or network being monitored.
- **Automation**—Automates root cause analysis and log file analysis, streamlines diagnostic workflows, and provides self-healing and remediation capabilities.
- **Greater network visibility**—Provides advanced multidimensional analytics across network elements, giving you a clearer understanding of network behavior to establish operational benchmarks, improve resource planning, and minimize service downtime.
- **Intuitive graphical user interface**—Offers an intuitive web-based GUI for policy management and easy data consumption.
- **Open integration**—Lowers the barrier of entry for telemetry and analytics by providing open source data pipelines, notification capabilities, and third-party device support.
- **Multiple data collection methods**—Includes support for Junos Telemetry Interface (JTI), NETCONF, syslog, NetFlow, and SNMP.

Closed-Loop Automation

HealthBot offers closed-loop automation. The automation workflow can be divided into seven main steps (see [Figure 1 on page 17](#)):

1. **Define**—HealthBot provides tools for the user to define the health parameters of key network elements through customizable key performance indicators (KPIs), rules, and playbooks.
2. **Collect**—HealthBot collects rule-based telemetry data from multiple devices using various types of data transfer methods.
3. **Store**—HealthBot stores time-sensitive telemetry data in a time-series database (TSDB). This allows users to query, perform operations on, and write new data back to the database, days, or even weeks after initial storage.
4. **Analyze**—HealthBot analyzes telemetry data based on customizable KPIs, rules, and playbooks.
5. **Visualize**—HealthBot provides multiple ways for you to visualize the aggregated telemetry data through the HealthBot web UI to gain actionable and predictive insight into the health of your devices and overall network.
6. **Notify**—HealthBot notifies you through the HealthBot web UI and alarm notifications when problems in the network are detected.
7. **Act**—HealthBot performs user-defined actions to help resolve and proactively prevent network problems.

Figure 1: HealthBot Closed-loop Automation Workflow



Main Components of HealthBot

HealthBot consists of three main components:

- **Health Monitoring**—View an abstracted, hierarchical representation of device and network-level health, and define the health parameters of key network elements through customizable key performance indicators (KPIs), rules, and playbooks.
- **Root Cause Analysis**—Find the root cause of a device or network-level issue when HealthBot detects a problem with a network element.
- **Log File Analysis**—Analyze relevant system log messages by filtering out noise.

HealthBot Health Monitoring

The Challenge

With increasing data traffic generated by cloud-native applications and emerging technologies, service providers and enterprises need a network analytics solution to analyze volumes of telemetry data, offer

insights into overall network health, and produce actionable intelligence. While telemetry-based techniques have existed for years, the growing number of protocols, data formats, and key performance indicators (KPIs) from diverse networking devices has made data analysis complex and costly. Traditional CLI-based interfaces require specialized skills to extract business value from telemetry data, creating a barrier to entry for network analytics

The HealthBot Health Monitoring Solution

By aggregating and correlating raw telemetry data from multiple sources, the HealthBot health monitoring feature provides a multidimensional view of network health that reports current status, as well as projected threats to the infrastructure and its workloads.

Health status determination is tightly integrated with the HealthBot root cause analysis (RCA) application, which can make use of syslog log data received from the network and its devices. HealthBot health monitoring provides status indicators that alert you when, for example, a network resource is currently operating outside a user-defined performance policy, as well as risk analysis using historical trends to predict whether a resource may be unhealthy in the future. HealthBot health monitoring not only offers a fully customizable view of the current health of network elements, it also automatically initiates remedial actions based on predefined service level agreements (SLAs).

Defining the health of a network element, such as broadband network gateway (BNG), provider edge (PE), core, and leaf-spine, is highly contextual. Each element plays a different role in a network, with unique key performance indicators (KPIs) to monitor. Given that there is no single definition for network health across all use cases, HealthBot provides a highly customizable framework to allow you to define your own health profiles.

HealthBot Root Cause Analysis

The Challenge

In some cases, it can be challenging for a network operator to figure out what causes a Junos OS networking device to stop working properly. When this happens, the typical workflow to find the root cause of the network problem involves contacting a specialist from Juniper Networks, who would then troubleshoot and triage the unhealthy component based on knowledge built from years of experience. After completing this time-intensive assessment, the problem would then be reassigned to the relevant engineering team.

The HealthBot RCA Solution

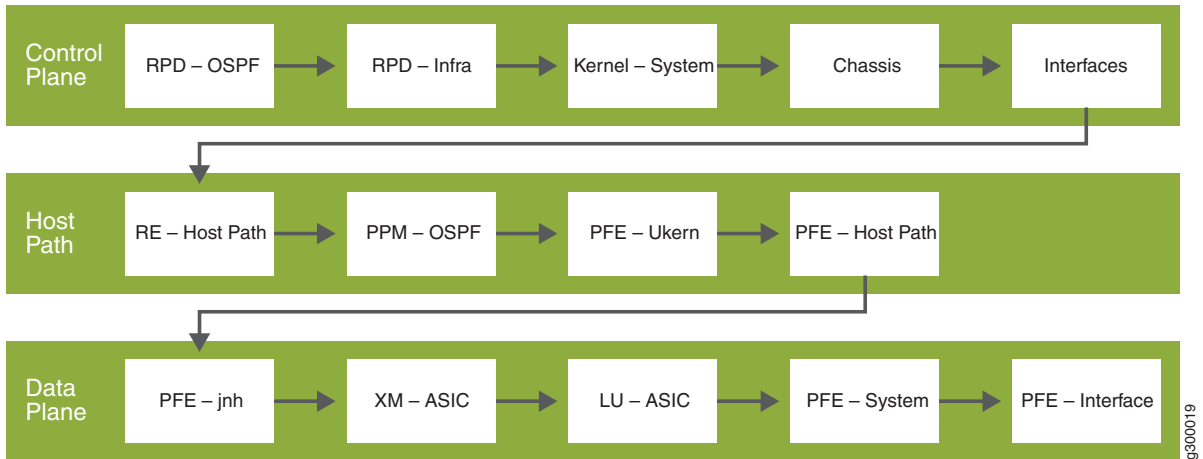
The purpose of the HealthBot root cause analysis (RCA) application is to simplify the process of finding the root cause of a network issue. HealthBot RCA captures the troubleshooting knowledge of, for example, the Juniper Networks specialists as part of a knowledge base in the form of HealthBot rules. These rules are evaluated either on demand by a specific trigger or periodically in the background to ascertain the health of a networking component, such as routing protocol, system, interface, or chassis, on the device.

To illustrate the benefits of HealthBot RCA, let us consider the problem of OSPF flapping.

[Figure 2 on page 19](#) highlights the workflow sequence involved in debugging OSPF flapping. If a network

operator or Juniper Networks specialist were assigned this troubleshooting task, he or she would need to perform manual debugging steps for each tile of the workflow sequence in order to find the root cause of the OSPF flapping. The HealthBot RCA application, on the other hand, delivers this expert service to you automatically as a bot. The RCA bot tracks all of the telemetry data collected by the HealthBot and translates the information into graphical status indicators (displayed in the HealthBot web UI) that correlate to different parts of the workflow sequence shown in [Figure 2 on page 19](#).

Figure 2: High-level workflow to debug OSPF-flapping



When configuring HealthBot, each tile of the workflow sequence shown in [Figure 2 on page 19](#) can be defined by one or more rules. For example, the RPD-OSPF tile could be defined as two rule conditions: one to check if "hello-transmitted" counters are incrementing and the other to check if "hello-received" counters are incrementing. Based on these user-defined rules, HealthBot provides status indicators, alarm notifications, and an alarm management tool through the web UI to inform and alert you of specific network conditions that could, for example, lead to OSPF flapping. By isolating a problem area in the workflow, HealthBot RCA proactively guides you in determining the appropriate corrective action to take to fix a pending issue or avoid a potential one.

HealthBot Log File Analysis

The Challenge

Networking devices can generate a lot of log messages, some of these messages are arcane and others create a lot of noise and clutter that drown out the more significant, meaningful messages. Network operators need an easy way to sort through and organize all of these log messages, as well as make sense of the information in order to take action, if necessary.

The HealthBot Log File Analysis Solution

Fully integrated with the HealthBot health monitoring and RCA features, HealthBot log file analysis can be implemented with the use of log patterns and pattern sets within the syslog ingest settings. The pattern sets can be applied to Rules to automatically filter out unnecessary log messages and help highlight only the relevant, actionable messages. Healthbot log file analysis consists of two main components:

1. An ingest engine that lets HealthBot receive syslog messages from networks and devices.
2. Pre-defined and customizable search patterns and pattern sets that can be applied to rules.

See *Syslog Ingest* for more information about syslog ingest.

RELATED DOCUMENTATION

| *HealthBot Getting Started Guide*

HealthBot Concepts

IN THIS SECTION

- [HealthBot Data Collection Methods | 21](#)
- [HealthBot Topics | 22](#)
- [HealthBot Rules - Basics | 23](#)
- [HealthBot Rules - Deep Dive | 25](#)
- [HealthBot Playbooks | 35](#)

HealthBot is a highly programmable telemetry-based analytics application. With it, you can diagnose and root cause network issues, detect network anomalies, predict potential network issues, and create real-time remedies for any issues that come up.

To accomplish this, network devices and HealthBot have to be configured to send and receive large amounts of data, respectively. Device configuration is covered throughout this and other sections of the guide.

Configuring HealthBot, or any application, to read and react to incoming telemetry data requires a language that describes several elements that are specific to the systems and data under analysis. This type of language is called a Domain Specific Language (DSL), i.e., a language that is specific to one domain. Any DSL is built to help answer questions. For HealthBot, these questions are:

- Q: What components make up the systems that are sending data?

A: Network devices are made up of memory, cpu, interfaces, protocols and so on. In HealthBot, these are called “[HealthBot Topics](#)” on page 22.

- Q: How do we gather, filter, process, and analyze all of this incoming telemetry data?

A: HealthBot uses “[HealthBot Rules - Basics](#)” on page 23 that consist of information blocks called sensors, fields, variables, triggers, and more.

- Q: How do we determine what to look for?

A: It depends on the problem you want to solve or the question you want to answer. Healthbot uses “[HealthBot Playbooks](#)” on page 35 to create collections of specific rules and apply them to specific groups of devices in order accomplish specific goals. For example, part of the **system-kpis-playbook** can alert a user when system memory usage crosses a user-defined threshold.

This section covers these key concepts and more, which you need to understand before using HealthBot.

HealthBot Data Collection Methods

In order to provide visibility into the state of your network devices, HealthBot first needs to collect their telemetry data and other status information. It does this using sensors.

HealthBot supports sensors that “push” data from the device to HealthBot and sensors that require HealthBot to “pull” data from the device using periodic polling.

Data Collection - 'Push' Model

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 HealthBot 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 \(JTI\)](#).

HealthBot currently supports four 'push' ingest types.

- Native GPB
- NetFlow

- OpenConfig
- Syslog

These push-model data collection—or *ingest*—methods are explained in detail in the *HealthBot Data Ingest Guide*.

Data Collection - 'Pull' Model

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. With the 'pull' model, HealthBot requests data from network devices at periodic intervals.

HealthBot currently supports two 'pull' ingest types.

- iAgent (CLI/NETCONF)
- SNMP

These pull-model data collection—or *ingest*—methods are explained in detail in the *HealthBot Data Ingest Guide*.

HealthBot Topics

Network devices are made up of a number of components and systems from CPUs and memory to interfaces and protocol stacks and more. In HealthBot, a topic is the construct used to address those different device components. The **Topic** block is used to create name spaces that define what needs to be modeled. Each **Topic** block is made up of one or more **Rule** blocks which, in turn, consist of the **Field** blocks, **Function** blocks, **Trigger** blocks, etc. See [“HealthBot Rules - Deep Dive” on page 25](#) for details. Each rule created in HealthBot must be part of a topic. Juniper has curated a number of these system components into a list of **Topics** such as:

- chassis
- class-of-service
- external
- firewall
- interfaces
- kernel
- linecard
- logical-systems
- protocol

- routing-options
- security
- service
- system

You can create sub-topics underneath any of the Juniper topic names by appending `.<sub-topic>` to the topic name. For example, `kernel.tcpip` or `system.cpu`.

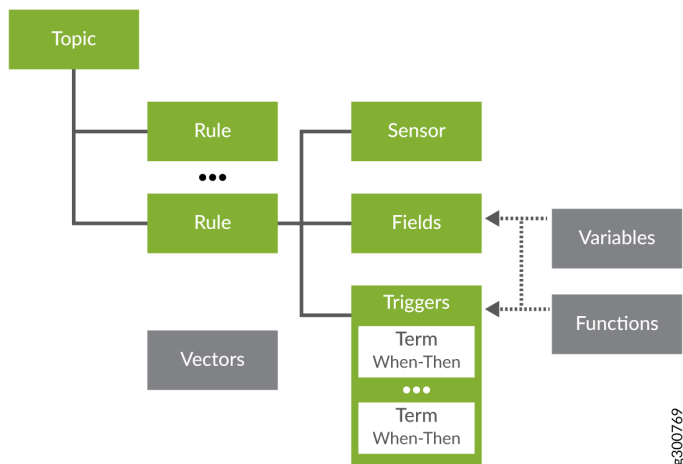
Any pre-defined rules provided by Juniper fit within one of the Juniper topics with the exception of *external*. The *external* topic is reserved for user-created rules. In the HealthBot web GUI, when you create a new rule, the **Topics** field is automatically populated with the *external* topic name.

HealthBot Rules - Basics

HealthBot's primary function is collecting and reacting to telemetry data from network devices. Defining how to collect the data, and how to react to it, is the role of a *rule*.

HealthBot ships with a set of default rules, which can be seen on the **Configuration > Rules** page of the HealthBot GUI, as well as in GitHub in the [healthbot-rules](#) repository. You can also create your own rules.

The structure of a HealthBot rule looks like this:



To keep rules organized, HealthBot organizes them into *topics*. Topics can be very general, like **system**, or they can be more granular, like **protocol.bgp**. Each topic contains one or more rules.

As described above, a *rule* contains all the details and instructions to define how to collect and handle the data. Each rule contains the following required elements:

- The *sensor* defines the parameters for collecting the data. This typically includes which data collection method to use (as discussed above in [“HealthBot Data Collection Methods” on page 21](#)), some guidance on which data to ingest, and how often to push or pull the data. In any given rule, a sensor can be defined directly within the rule or it can be referenced from another rule.
 - Example: Using the SNMP sensor, poll the network device every 60 seconds to collect all the device data in the Juniper SNMP MIB table [jnxOperatingTable](#).
- The sensor typically ingests a large set of data, so *fields* provide a way to filter or manipulate that data, allowing you to identify and isolate the specific pieces of information you care about. Fields can also act as placeholder values, like a static threshold value, to help the system perform data analysis.
 - Example: Extract, isolate, and store the [jnxOperating15MinLoadAvg](#) (CPU 15-minute average utilization) value from the SNMP table specified above in the sensor.
- *Triggers* periodically bring together the fields with other elements to compare data and determine current device status. A trigger includes one or more 'when-then' statements, which include the parameters that define how device status is visualized on the health pages.
 - Example: Every 90 seconds, check the CPU 15min average utilization value, and if it goes above a defined threshold, set the device's status to red on the device health page and display a message showing the current value.

The rule can also contain the following optional elements:

- *Vectors* allow you to leverage existing elements to avoid the need to repeatedly configure the same elements across multiple rules.
 - Examples: A rule with a configured sensor, plus a vector to a second sensor from another rule; a rule with no sensors, and vectors to fields from other rules
- *Variables* can be used to provide additional supporting parameters needed by the required elements above.
 - Examples: The string “ge-0/0/0”, used within a field collecting status for all interfaces, to filter the data down to just the one interface; an integer, such as “80”, referenced in a field to use as a static threshold value
- *Functions* allow you to provide instructions (in the form of a Python script) on how to further interact with data, and how to react to certain events.
 - Examples: A rule that monitors input and output packet counts, using a function to compare the count values; a rule that monitors system storage, invoking a function to cleanup temp and log files if storage utilization goes above a defined threshold

NOTE: Rules, on their own, don't actually do anything. To make use of rules you need to add them to [“HealthBot Playbooks” on page 35](#).

HealthBot Rules - Deep Dive

IN THIS SECTION

- Rules | 25
- Sensors | 28
- Fields | 28
- Vectors | 30
- Variables | 30
- Functions | 31
- Triggers | 31
- Tagging | 34
- Rule Properties | 35

A rule is a package of components, or blocks, needed to extract specific information from the network or from a Junos device. **Rules** conform to a specifically tailored domain specific language (DSL) for analytics applications. The DSL is designed to allow rules to capture:

- The minimum set of input data that the rule needs to be able to operate
- The minimum set of telemetry sensors that need to be configured on the device(s)
- The fields of interest from the configured sensors
- The reporting or polling frequency
- The set of triggers that operate on the collected data
- The conditions or evaluations needed for triggers to kick in
- The actions or notifications that need to be performed when a trigger kicks in

The details around rules, topics and playbooks are presented in the following sections.

Rules

Rules are meant to be free of any hard coding. Think of threshold values; If a threshold is hard coded, there is no easy way to customize it for a different customer or device that has different requirements. Therefore, rules are defined using parameterization to set the default values. This allows the parameters to be left at default or be customized by the operator at the time of deployment. Customization can be done at the device group or individual device level while applying the [HealthBot Playbooks on page 35](#) in which the individual rules are contained.

Rules that are device-centric are called device rules. Device components such as chassis, system, linecards, and interfaces are all addressed as [HealthBot Topics on page 22](#) in the rule definition. Generally, device rules make use of sensors on the devices.

Rules that span multiple devices are called network rules. Network rules:

- must have a rule-frequency configured
- must not contain sensors
- cannot be mixed with device rules in a playbook

To deploy either type of rule, include the rule in a playbook and then apply the playbook to a device group or network group.

NOTE: HealthBot comes with a set of pre-defined rules.

Not all of the blocks that make up a rule are required for every rule. Whether or not a specific block is required in a rule definition depends on what sort of information you are trying to get to. Additionally, some rule components are not valid for network rules. [Table 3 on page 26](#) lists the components of a rule and provides a brief description of each one.

Table 3: Rule Components

Block	What it Does	Required in Device Rules?	Valid for Network Rules?
“Sensors” on page 28	<p>The Sensors block is like the access method for getting at the data. There are multiple types of sensors available in HealthBot: OpenConfig, Native GPB, iAgent, SNMP, and syslog.</p> <p>It defines what sensors need to be active on the device in order to get to the data fields on which the triggers eventually operate. Sensor names are referenced by the Fields.</p> <p>OpenConfig and iAgent sensors require that a frequency be set for push interval or polling interval respectively. SNMP sensors also require you to set a frequency.</p>	No—Rules can be created that only use a field reference from another rule or a vector with references from another rule. In these cases, rule-frequency must be explicitly defined.	No

Table 3: Rule Components (continued)

Block	What it Does	Required in Device Rules?	Valid for Network Rules?
Fields on page 28	The source for the Fields block can be a pointer to a sensor, a reference to a field defined in another rule, a constant, or a formula. The field can be a string, integer or floating point. The default field type is string.	Yes- Fields contain the data on which the triggers operate. Starting in HealthBot release 3.1.0, regular fields and key-fields can be added to rules based on conditional tagging profiles. See the “Tagging” on page 34 section below.	Yes
“Vectors” on page 30	The Vectors block allows handling of lists, creating sets, and comparing elements amongst different sets. A vector is used to hold multiple values from one or more fields.	No	Yes
“Variables” on page 30	The Variables block allows you to pass values into rules. Invariant rule definitions are achieved through mustache-style templating like <code>{{<placeholder-variable>}}</code> . The placeholder-variable value is set in the rule by default or can be user-defined at deployment time.	No	No
“Functions” on page 31	The Functions block allows you to extend fields, triggers, and actions by creating prototype methods in external files written in languages like python. The functions block includes details on the file path, method to be accessed, and any arguments, including argument description and whether it is mandatory.	No	No
“Triggers” on page 31	The Triggers block operates on fields and are defined by one or more <i>Terms</i> . When the conditions of a Term are met, then the action defined in the Term is taken. By default, triggers are evaluated every 10 seconds, unless explicitly configured for a different frequency. By default, all triggers defined in a rule are evaluated in parallel.	Yes- Triggers enable rules to take action.	Yes
“Rule Properties” on page 35	The Rule Properties block allows you to specify metadata for a HealthBot rule, such as hardware dependencies, software dependencies, and version history.	No	Yes

Sensors

When defining a sensor, you must specify information such as sensor name, sensor type and data collection frequency. As mentioned in [Table 3 on page 26](#), sensors can be one of the following:

- **OpenConfig**—For information on OpenConfig JTI sensors, see the *Junos Telemetry Interface User Guide*.
- **Native GPB**—For information on Native GPB JTI sensors, see the *Junos Telemetry Interface User Guide*.
- **iAgent**—The iAgent sensors use NETCONF and YAML-based PyEZ tables and views to fetch the necessary data. Both structured (XML) and unstructured (VTY commands and CLI output) data are supported. For information on Junos PyEZ, see the [Junos PyEZ Documentation](#).
- **SNMP**—Simple Network Management Protocol.
- **syslog**—system log
- **BYOI**—Bring your own ingest - Allows you to define your own ingest types.
- **Flow**—NetFlow traffic flow analysis protocol
- **sFlow**—sFlow packet sampling protocol

When different rules have the same sensor defined, only one subscription is made per sensor. A key, consisting of *sensor-path* for OpenConfig and Native GPB sensors, and the tuple of *file* and *table* for iAgent sensors is used to identify the associated rule.

When multiple sensors with the same *sensor-path* key have different frequencies defined, the lowest frequency is chosen for the sensor subscription.

Fields

There are four types of field sources, as listed in [Table 3 on page 26](#). [Table 4 on page 29](#) describes the four field ingest types in more detail.

Table 4: Field Ingest Type Details

Field Type	Details
Sensor	<p>Subscribing to a sensor typically provides access to multiple columns of data. For instance, subscribing to the OpenConfig interface sensor provides access to a bunch of information including counter related information such as:</p> <p>/interfaces/counters/tx-bytes, /interfaces/counters/rx-bytes, /interfaces/counters/tx-packets, /interfaces/counters/rx-packets, /interfaces/counters/oper-state, etc.</p> <p>Given the rather long names of paths in OpenConfig sensors, the Sensor definition within Fields allows for aliasing, and filtering. For single-sensor rules, the required set of Sensors for the Fields table are programmatically auto-imported from the raw table based on the triggers defined in the rule.</p>
Reference	<p>Triggers can only operate on Fields defined within that rule. In some cases, a Field might need to reference another Field or Trigger output defined in another Rule. This is achieved by referencing the other field or trigger and applying additional filters. The referenced field or trigger is treated as a stream notification to the referencing field. References aren't supported within the same rule.</p> <p>References can also take a time-range option which picks the value, if available, from the time-range provided. Field references must always be unambiguous, so proper attention must be given to filtering the result to get just one value. If a reference receives multiple data points, or values, only the latest one is used. For example, if you are referencing a the values contained in a field over the last 3 minutes, you might end up with 6 values in that field over that time-range. HealthBot only uses the latest value in a situation like this.</p>
Constant	<p>A field defined as a constant is a fixed value which cannot be altered during the course of execution. HealthBot Constant types can be strings, integers, and doubles.</p>
Formula	<p>Raw sensor fields are the starting point for defining triggers. However, Triggers often work on derived fields defined through formulas by applying mathematical transformations.</p> <p>Formulas can be pre-defined or user-defined (UDF). Pre-defined formulas include: Min, Max, Mean, Sum, Count, Rate of Change, Elapsed Time, Standard Deviation, Microburst, Dynamic Threshold, Anomaly Detection, Outlier Detection, and Predict.</p> <p>Some pre-defined formulas can operate on time ranges in order to work with historical data. If a time range is not specified, then the formula works on current data, specified as <i>now</i>.</p>

Vectors

Vectors are useful in helping to gather multiple elements into a single rule. For example, using a vector you could gather all of the interface error fields. The syntax for Vector is:

```
vector <vector-name>{
  path [$field-1 $field-2 .. $field-n];
  filter <list of specific element(s) to filter out from vector>;
  append <list of specific element(s) to be added to vector>;
}
```

\$field-n can be field of type reference.

The fields used in defining vectors can be direct references to fields defined in other rules:

```
vector <vector-name>{
  path [/device-group[device-group-name=<device-group>]\
/device[device-name=<device>]/topic[topic-name=<topic>]\
/rule[rule-name=<rule>]/field[<field-name>=<field-value>]\
AND|OR ...]/<field-name> ...];
  filter <list of specific element(s) to filter out from vector>;
  append <list of specific element(s) to be added to vector>;
}
```

This syntax allows for optional filtering through the <field-name>=<field-value> portion of the construct. Vectors can also take a time-range option that picks the values from the time-range provided. When multiple values are returned over the given time-range, they are all selected as an array.

The following pre-defined formulas are supported on vectors:

- **unique @vector1**—Returns the unique set of elements from vector1
- **@vector1 and @vector2**—Returns the intersection of unique elements in vector1 and vector2.
- **@vector1 or @vector2**—Returns the total set of unique elements in the two vectors.
- **@vector1 unless @vector2**—Returns the unique set of elements in vector-1, but not in vector-2

Variables

Variables are defined during rule creation on the **Variables** page. This part of variable definition creates the default value that gets used if no specific value is set in the device group or on the device during

deployment. For example, the **check-interface-status** rule has one variable called **interface_name**. The value set on the **Variables** page is a regular expression (regex), `.*`, that means all interfaces.

If applied as-is, the **check-interface-status** rule would provide interface status information about all the interfaces on all of the devices in the device group. While applying a playbook that contains this rule, you could override the default value at the device group or device level. This allows you flexibility when applying rules. The order of precedence is device value overrides device group value and device group value overrides the default value set in the rule.

BEST PRACTICE: It is highly recommended to supply default values for variables defined in device rules. All Juniper-supplied rules follow this recommendation. Default values must not be set for variables defined in network rules.

Functions

Functions are defined during rule creation on the **Functions** tab. Defining a function here allows it to be used in **Formulas** associated with **Fields** and in the **When** and **Then** sections of **Triggers**. Functions used in the when clause of a trigger are known as user-defined functions. These must return true or false. Functions used in the then clause of a trigger are known as user-defined actions.

Triggers

Triggers play a pivotal role in HealthBot rule definitions. They are the part of the rule that determines if and when any action is taken based on changes in available sensor data. Triggers are constructed in a when-this, then-that manner. As mentioned earlier, trigger actions are based on **Terms**. A **Term** is built with *when* clauses that watch for updates in field values and *then* clauses that initiate some action based on what changed. Multiple **Terms** can be created within a single trigger.

Evaluation of the *when* clauses in the **Terms** starts at the top of the list of terms and proceeds to the bottom. If a *term* is evaluated and no match is made, then the next *term* is evaluated. By default, evaluation proceeds in this manner until either a match is made or the bottom of the list is reached without a match.

Pre-defined operators that can be used in the *when* clause include:

NOTE: For evaluated equations, the left-hand side and right-hand side of the equation are shortened to LHS and RHS, respectively in this document.

- *greater-than*—Used for checking if one value is greater than another.
 - Returns: True or False

- Syntax: `greater-than <LHS> <RHS> [time-range <range>]`
- Example: `//Memory > 3000 MB in the last 5 minutes`
`when greater-than $memory 3000 time-range 5m;`
- *greater-than-or-equal-to*—Same as *greater-than* but checks for greater than or equal to (`>=`)
- *less-than*
 - Returns: True or False
 - Syntax: `less-than <LHS> <RHS> [time-range <range>]`
 - Example: `//Memory < 6000 MB in the last 5 minutes`
`when less-than $memory 6000 time-range 5m;`
- *less-than-or-equal-to*—Same as *less-than* but checks for less than or equal to (`<=`)
- *equal-to*—Used for checking that one value is equal to another value.
 - Returns: True or False
 - Syntax: `equal-to <LHS> <RHS> [time-range <range>]`
 - Example: `//Queue's buffer utilization % == 0`
`when equal-to $buffer-utilization 0;`
- *not-equal-to*—Same as *equal-to* but checks for negative condition (`!=`)
- *exists*—Used to check if some value exists without caring about the value itself. Meaning that some value should have been sent from the device.
 - Returns: True or False
 - Syntax: `exists <$var> [time-range <range>]`
 - Example: `//Has the device configuration changed?`
`when exists $netconf-data-change`
- *matches-with (for strings & regex)*—Used to check for matches on strings using Python regex operations. See [Python Regular Expressions](#) for details.

NOTE: LHS, or left hand side, is the string in which we are searching; RHS, or right hand side, is the match expression. Regular expressions can only be used in RHS.

- Returns: True or False
- Syntax: `matches-with <LHS> <RHS> [time-range <range>]`
- Example: `//Checks that ospf-neighbor-state has been UP for the past 10 minutes`

when matches-with \$ospf-neighbor-state “^UP\$” time-range 10m;

- *does-not-match-with* (for strings & regex)–Same as *matches-with* but checks for negative condition
- *range*–Checks whether a value, X, falls within a given range such as minimum and maximum (min <= X <= max)
 - Returns: True or False
 - Syntax: range <\$var> min <minimum value> max <maximum value> [time-range <range>]
 - Example: **//Checks whether memory usage has been between 3000 MB and 6000 MB in the last 5 minutes**

when range \$mem min 3000 max 6000 time-range 5m;

- *increasing-at-least-by-value*–Used to check whether values are increasing by at least the minimum acceptable rate compared to the previous value. An optional parameter that defines the minimum acceptable rate of increase can be provided. The minimum acceptable rate of increase defaults to 1 if not specified.
 - Returns: True or False
 - Syntax:

increasing-at-least-by-value <\$var> [increment <minimum value of increase between successive points>]

increasing-at-least-by-value <\$var> [increment <minimum value of increase between successive points>] time-range <range>
 - Example: **Checks that the ospf-tx-hello has been increasing steadily over the past 5 minutes.**

when increasing-at-least-by-value \$ospf-tx-hello increment 10 time-range 5m;

- *increasing-at-most-by-value*–Used to check whether values are increasing by no more than the maximum acceptable rate compared to the previous value. An optional parameter that defines the maximum acceptable rate of increase can be provided. The maximum acceptable rate of increase defaults to 1 if not specified.
 - Returns: True or False
 - Syntax:

increasing-at-most-by-value <\$var> [increment <maximum value of increase between successive points>]

increasing-at-most-by-value <\$var> [increment <maximum value of increase between successive points>] time-range <range>
 - Example: **Checks that the error rate has not increased by more than 5 in the past 5 minutes.**

when increasing-at-most-by-value \$error-count increment 5 time-range 5m;

- *increasing-at-least-by-rate*—Used for checking that rate of increase between successive values is at least given rate. Mandatory parameters include the value and time-unit, which together signify the minimum acceptable rate of increase.

- Returns: True or False

- Syntax:

This syntax compares current value against previous value ensuring that it increases at least by value rate.

`increasing-at-least-by-rate <$var> value <minimum value of increase between successive points> per <second|minute|hour|day|week|month|year> [time-range <range>]`

This syntax compares current value against previous value ensuring that it increases at least by percentage rate

`increasing-at-least-by-rate <$var> percentage <percentage> per <second|minute|hour|day|week|month|year> [time-range <range>]`

- Example: **Checks that the ospf-tx-hello has been increasing strictly over the past five minutes.**

when increasing-at-least-by-rate \$ospf-tx-hello value 1 per second time-range 5m;

- *increasing-at-most-by-rate*—Similar to *increasing-at-least-by-rate*, except that this checks for decreasing rates.

Using these operators in the *when* clause, creates a function known as a user-defined condition. These functions should always return true or false.

If evaluation of a *term* results in a match, then the action specified in the *Then* clause is taken. By default, processing of terms stops at this point. You can alter this flow by enabling the **Evaluate next term** button at the bottom of the *Then* clause. This causes HealthBot to continue *term* processing to create more complex decision-making capabilities like when-this and this, then that.

The following is a list of pre-defined actions available for use in the *Then* section:

- next
- status

Tagging

Starting with Release 3.1.0, HealthBot supports tagging. Tagging allows you to insert fields, values, and keys into a HealthBot rule when certain conditions are met. See [“Healthbot Tagging” on page 36](#) for details.

Rule Properties

The **Rule Properties** block allows you to specify metadata for a HealthBot rule, such as hardware dependencies, software dependencies, and version history. This data can be used for informational purposes or to verify whether or not a device is compatible with a HealthBot rule.

HealthBot Playbooks

In order to fully understand any given problem or situation on a network, it is often necessary to look at a number of different system components, topics, or key performance indicators (KPIs). HealthBot operates on playbooks, which are collections of rules for addressing a specific use case. **Playbooks** are the HealthBot element that gets applied, or run, on your device groups or network groups.

HealthBot comes with a set of pre-defined **Playbooks**. For example, the system-KPI playbook monitors the health of system parameters such as system-cpu-load-average, storage, system-memory, process-memory, etc. It then notifies the operator or takes corrective action in case any of the KPIs cross pre-set thresholds. Following is a list of Juniper-supplied Playbooks.

- bgp-session-stats
- route-summary-playbook
- lldp-playbook
- interface-kpis-playbook
- system-kpis-playbook
- linecard-kpis-playbook
- chassis-kpis-playbook

You can create a playbook and include any rules want in it. You apply these playbooks to device groups. By default, all rules contained in a Playbook are applied to all of the devices in the device group. There is currently no way to change this behavior.

If your playbook definition includes network rules, then the playbook becomes a network playbook and can only be applied to network groups.

Release History Table

Release	Description
3.1.0	Starting in HealthBot release 3.1.0, regular fields and key-fields can be added to rules based on conditional tagging profiles.
3.1.0	Starting with Release 3.1.0, HealthBot supports tagging.

RELATED DOCUMENTATION

| [HealthBot Rules and Playbooks](#) | 118

Healthbot Tagging

IN THIS SECTION

- [Overview](#) | 36

Overview

Tagging allows you to insert fields, values, and keys into a HealthBot rule when certain conditions are met.

Tagging makes use of profiles to set the conditions, define the new fields and keys, and insert values into those fields after creation. One example of what you can do with this is simple application identification based on source-port, destination-port, and protocol of traffic seen in a NetFlow stream. The table below shows a small example of what we are talking about.

source-port	destination-port	protocol	derived-application
2541	Any	6 (TCP)	NetChat
Any	2541	6 (TCP)	
1755	Any	17 (UDP)	MS-streaming
Any	830	6 (TCP)	netconf-ssh
7802	Any	17 (UDP)	vns-tp

In the table above, we use three existing fields in a NetFlow stream to guess the application traffic in the stream. We then create a new field called *derived-application* and populate it based on the values seen in the traffic.

Another example is to help fine tune machine learning (ML) algorithms in HealthBot. If we are monitoring traffic flow for a sports streaming service, it would be helpful to tag traffic that occurs at specific times,

like the start and end of a high-profile football game. Even without ML, you could adjust traffic thresholds based on the timing of special events, thereby enhancing the customer experience.

HealthBot applies tagging profiles as part of ingest settings. This allows the tags to be added to the incoming data before HealthBot does a lot of processing, thereby allowing you to include your tags and values in calculations that you perform on incoming data.

The rest of this document describes how tagging is implemented in HealthBot and provides a couple of examples.

HealthBot Tagging Terminology

Since HealthBot configuration mimics the hierarchical method used by Junos OS, we can display HealthBot configuration elements as pseudo-config as shown below. This view shows how the elements of a tagging profile are named and how they are related to each other.

```
healthbot {
  ingest-settings {
    data-enrichment {
      tagging-profile <tagging-profile-name> {
        policy <policy-name> {
          rules [ List of Rules ];
          term <term-name1> {
            when {
              <condition1>
              <condition2>
            }
            then {
              add-field <field-name1> {
                value <field-value1>;
                type <field-type>;
              }
              add-field <field-name2> {
                value <field-value2>;
                type <field-type>;
              }
              add-key <key-field-name> {
                value <key-field-value>;
              }
            }
          }
          term <term-name1> {
            then {
              add-field <field-name> {
                value <field-value>;
              }
            }
          }
        }
      }
    }
  }
}
```

```

    type <field-type>;
  }
}
}
}
}
}
}
}
}
}
}
}
}

```

In the following sections we define the terms used within a tagging profile and provide some usage notes for each term.

Policy

A policy is the top-level element in a tagging profile. There can be multiple policies list used to name the policies in use within a tagging profile. Multiple policies can exist within a single profile, each with its own rules and terms.

Usage Notes:

Defining multiple policies within a single profile allows you to define terms for each rule in one profile rather than having to create one profile for each rule.

Rules

In terms of tagging profiles, a rule is any defined HealthBot rule. The rule element in a tagging profile is a list element. The rule or rules included in the list (*[rule1, rule2]*) get the tagging profile, and more specifically, the specific policy within the profile applied to them.

Usage Notes:

There are many ways to describe the topic-name/rule-name requirement for the rules element. They are listed below:

- To name specific rules within a tagging profile, use the form: *topic-name/rule-name* where topic-name and rule-name are defined seen in **Configuration > Rules**. For example:
protocol.bgp/check-bgp-advertised-routes.
- Use an asterisk (*) with no other value or brackets to match all rules.
- Use python-based fnmatch patterns to select all rules within a specific topic, like **line-cards/***, or rules within all topics, like ***/line-card**. See [fnmatch – Unix filename pattern matching](#).

Terms

The term section of the tagging profile is where the match conditions are set and examined, and actions based on those matches are set and carried out. We set the conditions for match in a *when statement*. We set the actions to be taken upon completing a match in one or more *then statements*.

Usage Notes:

- Each term can contain a *when* statement but it is not required.
- Each term must contain at least one *then* statement.
- Multiple terms can be set within a single policy.
- Terms are processed sequentially from top to bottom until a match is found. If a match is found, processing stops after completing the statements found in the *then* section; other terms, if present, are not processed unless the *next* flag is enabled within the matched term. If the matched term has the next flag enabled, then subsequent terms are processed in order.

When Statements

When statements define the match conditions that you specify. *When* statements ultimately resolve to true or false. A *term* can be defined that has no *when* statements. This equates to a default *term* wherein the match is assumed true and the subsequent *then* statement is carried out. Conversely, multiple conditions can be checked within one *when* statement.

For example, you can check whether traffic is using the UDP protocol on port 53, with source address of any, and a destination address of 8.8.8.8. Any traffic that matches all of these conditions can be tagged by the actions in the *then* statement contained in the same *term*.

If one or more of the conditions set forth in a *when* statement are not met, the statement is false and the *term* has failed to match; processing moves to the next *term*, if present.

Usage Notes:

When statements perform boolean operations on the received data to determine if it matches your criteria. The supported operations are listed below:

• Numeric Operations:

- equal-to
- not-equal-to
- greater-than
- greater-than-or-equal-to
- less-than
- less-than-or-equal-to

• String Operations:

- matches-with
- does-not-match-with
- **Time Operations:**
 - matches-with-scheduler

NOTE: The **matches-with-scheduler** option requires that a discreet scheduler be configured on the **Settings > System > Scheduler** page. The name of the scheduler can then be used in the **matches-with-scheduler** *when* statement

- **Go Language Expressions:**
 - eval *<simple-go-expression>*

For example: **eval a + b <= c.**

Then Statements

Then statements implement the tagging instructions you provide; but only after a complete match of the conditions set forth in a *when* statement contained in the same *term*. Each *term* defined must have at least one *then* statement. Each *then* statement must have one (or more) action(s) defined; the actions available in *then* statements are:

add-field—Adds a normal field to the rule(s) listed in the rule section. Multiple fields can be added within a *then* statement. The add-field action requires that you also define the kind of field you are adding with the *field-type* parameter:

- string
- integer
- float

NOTE: If you do not define a field type, the new field gets added with the default field-type of string.

add-key—Adds a key field with the data type of string to the rule(s). Added key fields are indexed and searchable just like any other key field.

Usage Notes:

As mentioned above, the *next* flag can be set within a *then* statement. When this flag is set, the next term in the policy gets evaluated if all of the conditions of the current term match.

How It Works

As mentioned previously, tagging profiles are created (in the HealthBot GUI) at **Settings > Ingest > Tagging Profile**. Since one or more rules are defined within each profile, you can probably guess that the profile is applied when one of the named rules is added to a playbook and applied to a device-group. This applies the profile to all devices in the group.

Tagging profiles can be applied at the device-group or device level. If tagging profiles are configured at both device-group and device level on the same device, they are merged. Take the pseudo-configuration shown below:

```
device r0 {
    host r0;
    tagging-profile [ profile1 ]
}
device r1 {
    host r1
}
device-group core {
    devices [ r0 r1 ];
    tagging-profile [ profile2 ]
}
```

In the example above, we have:

- Device *r0* has tagging profile, *profile1*, assigned at the device level and tagging profile *profile2* assigned by its membership in device-group *core*.
- Device *r1* has only the tagging profile, *profile2*, assigned by its membership in device-group *core*.

Given this scenario, *profile1* and *profile2* are merged on device *r0*, while device *r1* only gets profile *profile2*.

If *profile1* and *profile2* both define the same fields, but the fields contain different values, then the value from *profile1* takes precedence because it is assigned directly to the device.

Caveats

- Fields and keys added using tagging profiles cannot be used within periodic aggregation fields. This is because periodic aggregation must take place before any UDFCode functions (reference, vector, UDF, ML) are applied.
- Tagging-profiles can consist of only fields in add-key or add-field. Vectors cannot be added to a rule by a tagging-profile.
- Vector comparison operations cannot be used within tagging-profile terms. Only field Boolean operations are permitted.

- For tagging profile conditional operations within a *when* statement, the used field must be of type sensor, constant, reference.
- If the field used within tagging profile Boolean operations is of type reference, then this reference field must not depend on any user-defined-function or formula defined within same rule.

Examples

Static Tagging - No When Statement

```

healthbot {
  ingest-settings {
    data-enrichment {
      tagging-profile profile {
        policy policy1 {
          rules *;
          term term1 {
            then {
              add-key "tenant-id" {
                value tenant1;
              }
            }
          }
        }
      }
    }
  }
}

```

In the static tagging example above, the lack of a *when* statement means that any device that this profile is applied to will have the field *tenant-id* assigned with the value *tenant1*. The fields and values defined in this profile are assigned to all rules that are applied to a device or device-group because of the *** in the rules parameter.

The same profile, configured in the HealthBot GUI at **Settings > Ingest** on the Tagging Profile tab looks like [Figure 3 on page 43](#) below.

Figure 3: Add Static Tagging Profile

Add Tagging Profile

Name*
profile1

Policies

[Add Policy](#)

policy1 [Delete policy1](#)

Policy Name*
policy1

Rule(s)*

Terms

[Add Term](#)

Term: Term1 [Delete Term1](#)

Term Name*
Term1

When

[Add When](#)

Then*

Keys

Name*	Value*	
tenant-id	tenant1	<input type="text" value=""/> Delete

[Add Key](#)

Fields

[Add Field](#)

Evaluate next term

[CANCEL](#) [SAVE](#) [SAVE & DEPLOY](#)

Default Term Profile

```
healthbot {
  ingest-settings {
    data-enrichment {
      tagging-profile default-term1 {
```

```
policy policy1 {
  rules *;
  term term1 {
    when {
      matches-with "$field1" FPC1;
    }
    then {
      add-field "output-field" {
        value val1;
      }
    }
  }
  term term2 {
    then {
      add-field "output-field" {
        value val2;
      }
    }
  }
}
}
```

In the default term profile shown above, *term1* defines some criteria that must be met in order to add the field *output-field* with value *val1*. However, if there is no match for *term1*, then *term2* is processed and the field, *output-field*, gets the value *val2*. This definition results in the value, *val2*, being the default value for the field *output-field*.

The same profile, configured in the HealthBot GUI at **Settings > Ingest** on the Tagging Profile tab looks like [Figure 4 on page 45](#) below.

Figure 4: Default Tag

EDIT "default-term1"

Name*
default-term1

Policy Name*
policy1

Rules*
rule1

Terms

Term: term1

Term Name*
term1

When

rule1-key-with

Operator*
rule1-key-with

Left Operand	Right Operand
\$field1	FPC1

Then*

Keys

Name*	Value*	Type
output-field	val1	STRING

Fields

Term: term2

Term Name*
term2

When

Then*

Keys

Name*	Value*	Type
output-field	val2	STRING

Fields

Cancel SAVE SAVE & DEPLOY

Application Identification

Remember the table from the beginning of this section? If not, here it is again:

source-port	destination-port	protocol	derived-application
2541	Any	6 (TCP)	NetChat
Any	2541	6 (TCP)	
1755	Any	17 (UDP)	MS-streaming
Any	830	6 (TCP)	netconf-ssh
7802	Any	17 (UDP)	vns-tp

To create the derived-application field from the received data, you would need a tagging profile definition that looks like this:

```
healthbot {
  ingest-settings {
    data-enrichment {
      tagging-profile profile1 {
        policy policy1 {
          rules *;
          term term1 {
            when {
              matches-with "$source-port" "$netchat-source-port";

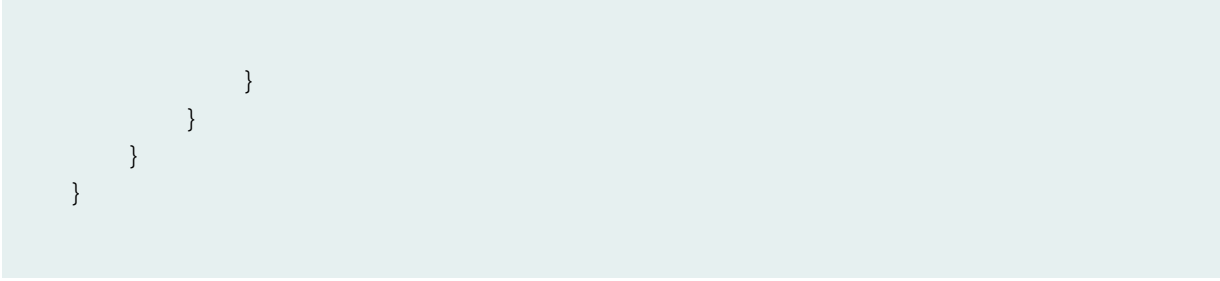
              matches-with "$protocol" "6 (TCP)";
            }
            then {
              add-key "application" {
                value netchat;
              }
            }
          }
          term term2 {
            when {
              matches-with "$protocol" "6 (TCP)";
              matches-with "$destination-port"
"$netchat-dest-port";
            }
            then {
```

```

        add-key "application" {
            value netchat;
        }
    }
}
term term3 {
    when {
        matches-with "$source-port"
"$ms-streaming-source-port";
        matches-with "$protocol" "17 (UDP)";
    }
    then {
        add-key "application" {
            value ms-streaming;
        }
    }
}
term term4 {
    when {
        matches-with "$source-port"
"$netconf-ssh-source-port";
        matches-with "$protocol" "6 (TCP)";
    }
    then {
        add-key "application" {
            value netconf-ssh;
        }
    }
}
term term5 {
    when {
        matches-with "$source-port" "$vns-tp-source-port";

        matches-with "$protocol" "17 (UDP)";
    }
    then {
        add-key "application" {
            value vns-tp;
        }
    }
}
}
}

```



The same profile, configured in the HealthBot GUI at **Settings > Ingest** on the Tagging Profile tab would be a very large image, so we are not showing it.

HealthBot Time Series Database (TSDB)

IN THIS SECTION

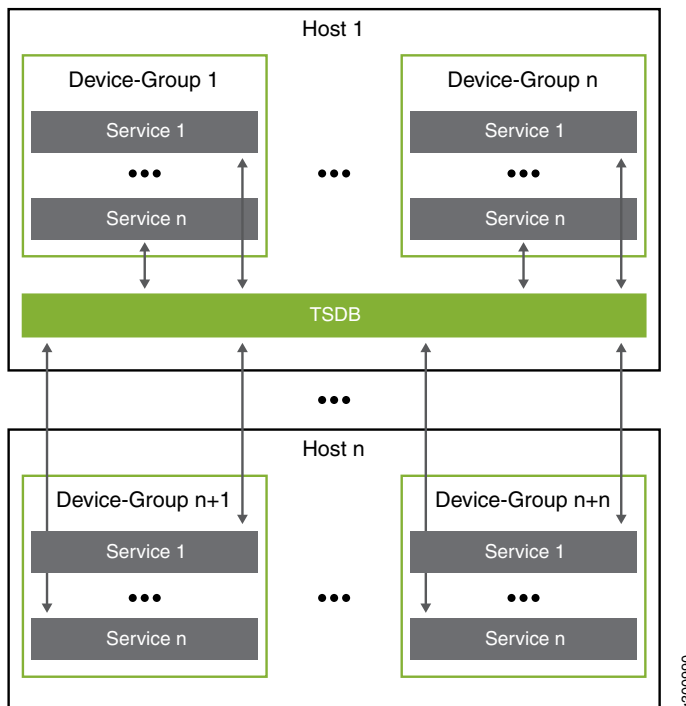
- [Historical Context | 48](#)
- [TSDB Improvements | 49](#)
- [Database Sharding | 50](#)
- [Database Replication | 51](#)
- [Database Reads and Writes | 52](#)
- [Manage TSDB Options in the HealthBot GUI | 53](#)
- [HealthBot CLI Configuration Options | 55](#)

HealthBot collects a lot of data through its various ingest methods. All of that data is time sensitive in some context. This is why HealthBot uses a time-series database (TSDB) to store and manage all of the information received from the various network devices. This topic provides an overview of the TSDB as well some guidance on managing it.

Historical Context

In HealthBot releases prior to 3.0.0, there was one TSDB instance regardless of whether you ran HealthBot as a single node or as a multi-node (Docker-compose) installation. [Figure 5 on page 49](#) shows a high-level view of what this looked like.

Figure 5: Single TSDB Instance - Prior to HealthBot Release 3.0.0



This arrangement left no room for scaling or redundancy of the TSDB. Without redundancy, there is no high availability (HA); A failure left you with no way to continue operation or restore missing data. Adding more Docker-compose nodes to this topology would only provide more HealthBot processing capability at the expense of TSDB performance.

TSDB Improvements

To address these issues and provide TSDB high availability (HA), three new TSDB elements are introduced in HealthBot Release 3.0.0, along with clusters of HealthBot nodes* for the other HealthBot microservices:

- [“Database Sharding” on page 50](#)– How many servers, or nodes, are available to store TSDB data and scale HealthBot?
- [“Database Replication” on page 51](#)– How many copies of your data do you want to keep?
- [“Database Reads and Writes” on page 52](#)– How is data written and read back from the TSDB? What happens when something goes wrong?

NOTE: *HealthBot uses Kubernetes for clustering its docker-based microservices across multiple physical or virtual servers (nodes). Kubernetes clusters consist of a primary node and multiple worker nodes. During the **healthbot setup** portion of HealthBot multinode installations, the installer asks for the IP addresses (or hostnames) of the Kubernetes primary node and worker nodes. You can add as many worker nodes to your setup as you need, based on the required replication factor for the TSDB databases. The number of nodes you deploy should be at least the same as the replication factor. (See the following sections for details).

For the purposes of this discussion, we refer to the Kubernetes worker nodes as HealthBot nodes. The primary node is not considered in this discussion.

Database Sharding

Database sharding refers to selectively storing data on certain nodes. This method of writing data to selected nodes distributes the data among available TSDB nodes and permits greater scaling since each TSDB instance then handles only a portion of the time series data from the devices.

To achieve sharding, HealthBot creates one database per device group/device pair and writes the resulting database to a specific (system determined) instance of TSDB hosted on one (or more) of the Healthbot nodes.

For example, say we have two devices, D1 and D2 and two device groups, G1 and G2. If D1 resides in groups G1 and G2, and D2 resides only in group G2, then we end up with 3 databases: G1:D1, G2:D1, and G2:D2. Each database is stored on its own TSDB instance on a separate HealthBot node as shown in [Figure 6 on page 51](#) below. When a new device is on-boarded and placed within a device group, HealthBot chooses a TSDB database instance on which to store that device/device-group data.

Figure 6: Distributed TSDB

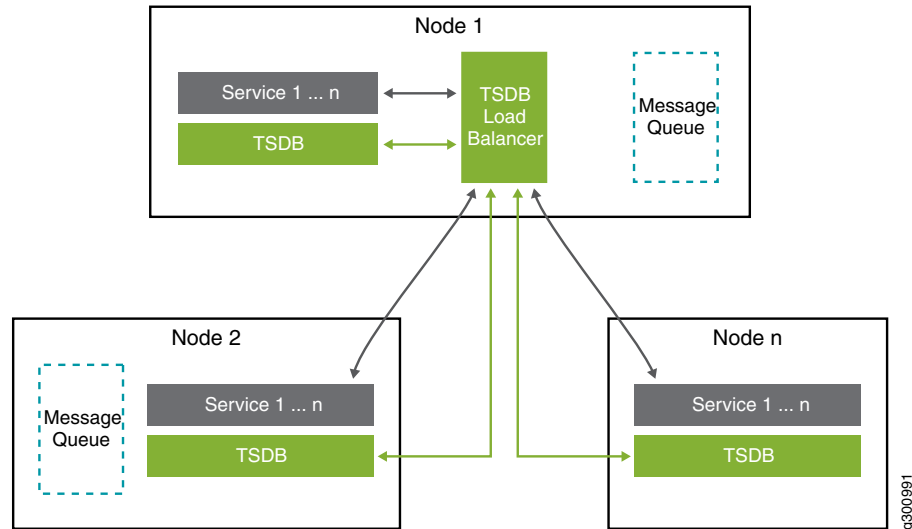


Figure 6 on page 51, above, shows 3 HealthBot nodes, each with a TSDB instance and other HealthBot services running.

NOTE:

- A maximum of 1 TSDB instance is allowed on any given HealthBot node. Therefore, a HealthBot node can have 0 or 1 TSDB instances at any time.
- A HealthBot node can be dedicated to running only TSDB functions. When this is done, no other HealthBot functions run on that node. This prevents other HealthBot functions from starving the TSDB instance of resources.
- We recommend that you dedicate nodes to TSDB to provide the best performance.
- HealthBot and TSDB nodes can be added to a running system using the HealthBot CLI.

Database Replication

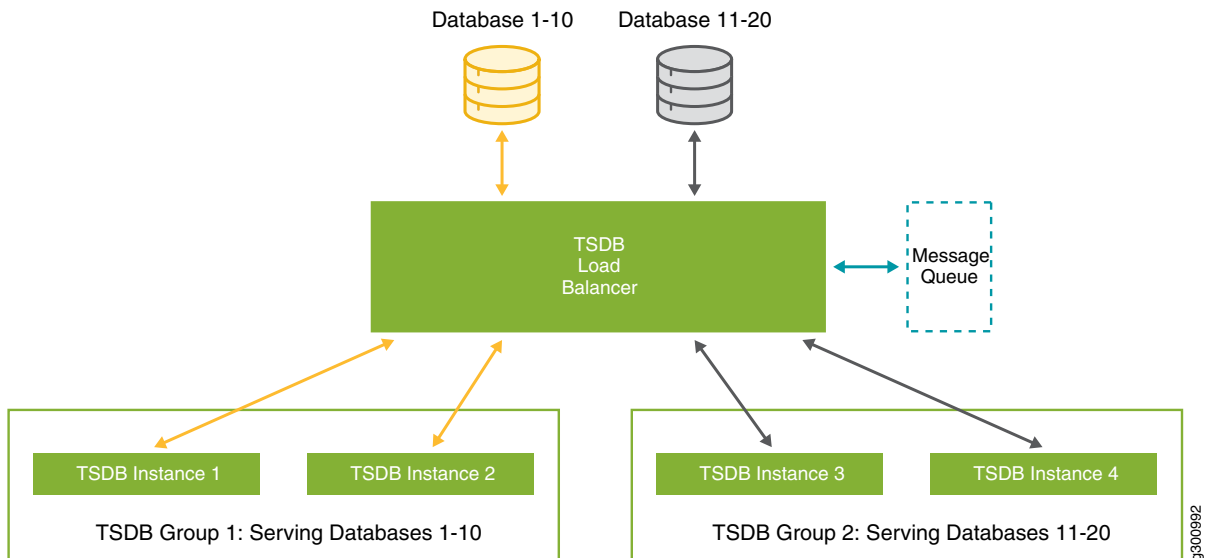
As with any other database system, replication refers to storing the data in multiple instances on multiple nodes. In HealthBot, we establish a replication factor to determine how many copies of the database are needed.

A replication factor of 1 only creates one copy of data, and therefore, provides no HA. When multiple HealthBot nodes are available and replication factor is set to 1, then only sharding is achieved.

The replication factor determines the minimum number of HealthBot nodes needed. A replication factor of 3 creates three copies of data, requires at least 3 HealthBot nodes, and provides HA. The higher the replication factor, the stronger the HA and the higher the resource requirements in terms of HealthBot nodes. If you want to scale your system further, you should add HealthBot nodes in exact multiples of the replication factor, or 3, 6, 9, etc.

Consider an example where, based on device/device-group pairing mentioned earlier, HealthBot has created 20 databases. The HealthBot system in question has a replication factor of 2 and has 4 nodes running TSDB. Based on this, two TSDB replication groups are created; in our example they are *TSDB Group 1* and *TSDB Group 2*. In [Figure 7 on page 52](#) below, the data from databases 1-10 is being written to TSDB instances 1 and 2 in TSDB group 1. Data from databases 11-20 is written to TSDB instances 3 and 4 in TSDB group 2. The outline around the TSDB instances represents a TSDB replication group. The size of the replication group is determined by the replication factor.

Figure 7: TSDB Databases



Database Reads and Writes

As shown in [Figure 6 on page 51](#), HealthBot can make use of a distributed messaging queue. In cases of performance problems or errors within a given TSDB instance, this allows for writes to the database to be performed in a sequential manner ensuring that all data is written in proper time sequence.

All HealthBot microservices use standardized database query (read) and write functions that can be used even if the underlying database system is changed at some point in the future. This allows for flexibility in growth and future changes. Other read and write features of the database system include:

- In normal operation, database writes are sent to all TSDB instances within a TSDB group.
- Database writes can be buffered up to 1GB per TSDB instance so that failed writes can be retried until successful.
- If problems persist and the buffer fills up, the oldest data is dropped in favor of new data.
- When buffering is active, database writes are performed sequentially so that new data cannot be written until the previous write attempts are successful.
- Database queries (reads) are sent to the TSDB instance which has reported the fewest write errors in the last 5 minutes. If all instances are performing equally, then the query is sent to a random TSDB instance in the required group.

Manage TSDB Options in the HealthBot GUI

TSDB options can be managed from within the HealthBot GUI. To do this, navigate to the **Settings > System** page from the left-nav bar, then select the **TSDB** tab from the left side of the page.

BEST PRACTICE: Adding, deleting, or dedicating TSDB nodes should be done during a maintenance window since some services will be restarted and the HealthBot GUI will likely be unresponsive while the TSDB work is performed.

In [Figure 8 on page 53](#) below, you can see that the current TSDB nodes and replication factor are shown.

Figure 8: TSDB System Settings

The screenshot displays the 'TsdB Settings' configuration page in the HealthBot GUI. The left sidebar shows the 'TSDB' tab selected. The main content area is titled 'TsdB Settings' and includes the following fields and controls:

- TsdB Nodes:** A dropdown menu showing two IP addresses: 10.102.70.82 and 10.102.70.200.
- Replication Factor:** A text input field containing the value 2.
- dedicate:** A toggle switch currently turned off.
- force:** A toggle switch currently turned off.
- Buttons:** Two blue buttons at the bottom: 'SAVE' and 'SAVE & DEPLOY'.

Working with TSDB Nodes

- In [Figure 8 on page 53](#) above, the field **TSDB Nodes** shows the currently defined TSDB instances in our HealthBot installation.
- The pull-down list of nodes under **TSDB Nodes** displays all currently available HealthBot nodes.
- Any operation you perform affects all the TSDB instances shown in the field. In this case, hosts **10.102.70.82** and **10.102.70.200**.
- You can delete a node from your HealthBot installation by clicking on the **X** next to the node IP or hostname that you want to remove. When you click **SAVE & DEPLOY**, that TSDB node is removed from HealthBot entirely.

NOTE: You cannot add HealthBot or TSDB nodes from the GUI in the current release of HealthBot. Refer to [Add a TSDB Node to HealthBot on page 55](#) to see the CLI procedure.

Change Replication Factor

- Increase or decrease the replication factor as needed.
- Click the **SAVE & DEPLOY** button to commit the change.

Dedicate a Node to TSDB

- Use the pull-down menu to select the node that you want to dedicate.
- Ensure that it is the only one in the **TSDB Nodes** list.
- Click the **dedicate** slider so that it activates (turns blue).
- Click the **SAVE & DEPLOY** button to commit the change.

Force Option

Normally, a failure in a TSDB instance can be overcome using the buffering methods described above.

In the event of a catastrophic failure of the server or storage hosting a TSDB instance, you can rebuild the server or damaged component. However, if the replication factor is 1, then the TSDB data for that instance is lost. In that case, you need to remove the failed TSDB node from HealthBot.

- Select the **X** next to the damaged node from the **TSDB Nodes** field.
- Click **SAVE & DEPLOY**.
- If a problem occurs and the removal is not successful, Click the **force** slider so that it activates (turns blue).

This tells the system to ignore any errors encountered while adjusting the TSDB settings.

- Click the **SAVE & DEPLOY** button to commit the change.

HealthBot CLI Configuration Options

The HealthBot CLI provides a means to add and delete TSDB nodes from the system and to change the replication factor as a result.

Add a TSDB Node to HealthBot

```
# set healthbot system time-series-database nodes <IP address or hostname>
dedicate <true or false>
```

or

```
#set healthbot system time-series-database nodes [ space-separated list of
IP addresses or hostnames] dedicate <true or false>
```

Manage the Replication Factor

```
# set healthbot system time-series-database replication-factor
<replication-factor>
```

Set the replication factor to a multiple of the number of TSDB nodes present in the system. If you have two TSDB nodes, set the replication factor at 2, 4, 6, etc.

Usage Notes

- HealthBot performs a ping to determine if the new node(s) is reachable. A warning is shown if the ping fails.
- The *dedicate* option specifies whether or not the TSDB nodes perform only TSDB functions.

RELATED DOCUMENTATION

| [HealthBot Installation Guide](#)

HealthBot Machine Learning (ML)

IN THIS SECTION

- [HealthBot Machine Learning Overview | 56](#)
- [Understanding HealthBot Anomaly Detection | 57](#)
- [Understanding HealthBot Outlier Detection | 59](#)
- [Understanding HealthBot Predict | 62](#)
- [HealthBot Rule Examples | 64](#)

HealthBot Machine Learning Overview

HealthBot uses machine learning to detect anomalies and outliers, and predict future device or network-level behavior. The machine learning-enabled HealthBot features include:

Anomaly Detection—Anomaly detection using the HealthBot Anomaly Detection algorithms involves comparison of new data points with data points collected from the same device during a specific learning period. HealthBot supports the following machine learning algorithms for anomaly detection:

- 3-sigma
- K-means
- Holt-Winters

Anomaly detection can be activated within HealthBot rules by setting a rule field's ingest type to formula, and then selecting anomaly detection. (**Configuration > Rules > Fields tab > Ingest Type > Formula**).

Outlier Detection—Outlier detection using the HealthBot Outlier Detection algorithms involves analysis of data from a collection of devices across your network during a specific learning period. HealthBot supports the following machine learning algorithms for outlier detection:

- Density-Based Spatial Clustering of Applications with Noise (DBSCAN)
- K-fold cross-validation using 3-sigma (k-fold/3-sigma)

Prediction—Prediction of future device or network-level behavior involves using the HealthBot median prediction machine learning algorithm or, the Holt-Winters prediction algorithm.

Starting with HealthBot Release 3.1.0, you can choose the Holt-Winters prediction algorithm from **Configuration > Rules > Fields > Ingest Type > Formula**.

Understanding HealthBot Anomaly Detection

IN THIS SECTION

- [Field | 57](#)
- [Algorithm | 58](#)
- [Learning period | 58](#)
- [Pattern periodicity | 59](#)

This section describes the input parameters associated with HealthBot rules configured to detect anomalies using Anomaly Detection algorithms. Once the machine learning models are built, they can be used in production to classify new data points as normal or abnormal. The accuracy of the results increases with larger amounts of data.

Field

To apply a machine learning algorithm, you must first define the numeric data field on which to apply the algorithm. For information on how to create a user-defined data field for a HealthBot rule, see the Fields section in the HealthBot User Guide.

Algorithm

The HealthBot Anomaly Detection algorithms include Holt-Winters, 3-sigma and k-means:

Holt-Winters—The Holt-Winters algorithm uses traffic entropy measurements and seasonal variations in traffic to detect anomalous traffic flowing through an interface. The seasonality aspect provides a means to de-emphasize normal increases and decreases in traffic that happen regularly over time intervals. For example, network traffic in an enterprise network could be expected to have a weekly seasonality since there would be significantly more traffic on the network during the work week than on the weekend.

Since Holt-Winters can predict a traffic drop starting on Friday evening, an anomaly might be triggered if traffic actually increased on Friday evening.

3-Sigma—The 3-sigma algorithm classifies a new data point as normal if it's within 3 standard deviations from the mean (average across all the data points in the data set). A new data point is classified as abnormal if it's outside this range.

K-means—The HealthBot k-means algorithm uses k-means clustering and other building blocks to create a machine learning model for classifying new data points as normal or abnormal:

- K-means clustering splits n data points into k groups (called clusters), where $k \leq n$. For HealthBot, k is set to 5 buckets.
- For forming the clusters, a 32-dimensional vector is considered for each point, thus taking into account the trend (not just the current point, but its past 31 historical values).
- Each cluster has a center called the centroid. A cluster centroid is the mean of a cluster (average across all the data points in the cluster).
- All new data points are added to a cluster, however, if a data point is considered too far away from the centroid, it is classified as abnormal.

Learning period

The learning period specifies the time range to collect data from which the algorithm uses to build the machine learning models. Supported units of time for learning period include: seconds, minutes, hours, days, weeks, and years. You must enter the plural form of the time unit with no space between the number and the unit. For example, a learning period of one hour must be entered as 1hours.

HealthBot builds machine learning models daily starting at midnight. For example, if the learning period is 5 days and triggered on 11th Feb 2019 00:00, then data collected from 6th Feb 2019 00:00 to 11th Feb 2019 00:00 is used by HealthBot to build the machine learning models. For the Holt-Winters prediction algorithm, the learning period must be at least twice the pattern periodicity to ensure there is enough of a pattern to learn.

Pattern periodicity

The pattern periodicity specifies the buckets for which data should be collected and used to build machine learning models. Each bucket of data represents a user-defined time period and a specific pattern of data. A minimum number of data points is required for a machine learning algorithm to build a model:

- 3-sigma requires at least 10 data points per bucket of data.
- K-means requires at least 32 data points per bucket of data.

Supported units of time for pattern periodicity include: minutes, hours, days, weeks, and months. You must enter the plural form of the time unit with no space between the number and the unit.

For example:

- If the pattern periodicity is 1 day (entered as 1days), the data for each day of the week has a specific pattern. HealthBot creates 7 buckets of data and 7 different models, one for each day of the week.
- If the pattern periodicity is 1 hour (entered as 1hours), regardless of the day, week, or month, the data for every hour has a specific pattern. HealthBot creates 24 buckets of data and 24 different models, one for each hour (00:00-00:59, 1:00-1:59, 2:00-2:59 ... 23:00-23:59) of the day.
- If the pattern periodicity is 1 day 1 hour (entered as 1days 1hours), the data for every hour of each day of the week has a specific pattern. HealthBot creates $7 * 24 = 168$ buckets of data and 168 different models. 24 buckets for Monday (1 for every hour), 24 buckets for Tuesday (1 for every hour), and so on. In this case, it doesn't matter from which month data is collected.

Understanding HealthBot Outlier Detection

IN THIS SECTION

- Dataset | 60
- Algorithm | 60
- Sigma coefficient (k-fold-3sigma only) | 62
- Sensitivity | 62
- Learning period | 62

This section describes the input parameters associated with HealthBot rules used for outlier detection algorithms. Once the machine learning models are built, they can be used in production to identify time series data sets as outliers. The accuracy of the results increases with larger amounts of data.

The results of the HealthBot outlier detection algorithm are stored in a table in the times series database. Each row in the table contains outlier detection output and metadata that is associated with a particular time series. You can use the information in the table to configure HealthBot rule triggers. Each column in the table is identified by a unique name that starts with the user-defined outlier detection field name. For example, you can use the *field-name-is-outlier* and *field-name-device* column names to configure a trigger that detects outliers and produces a message that indicates which specific device was determined to be the outlier. For more information, see the “Triggers” section of the [“HealthBot Outlier Detection Example” on page 73](#).

Dataset

For the outlier detection formula, input data is specified as a list of XPATHs from a variable. For information on how to create a user-defined variable for a HealthBot rule, see the [Variables](#) section in the Contrail HealthBot User Guide.

The following is an example of a list of XPATHs:

```
/device-group[device-group-name=DG0]/device[device-name=D0]/topic[topic-name=T0]/
rule[rule-name=R0]/field[re=RE[01] AND hostname=1.1.1.*]/re-memory,/
device-group[device-group-name=DG0]/device[device-name=D1]/topic[topic-name=T0]/
rule[rule-name=R0]/field[re=RE[01] AND hostname=1.1.1.*]/re-memory
```

This path list specifies that on devices D0 and D1 in device-group DG0, get re-memory from topic T0 rule R0, where the RE is either RE0 or RE1 and the hostname is in the 1.1.1.* block. This path allows for selecting data at the field-key level, which is necessary because different field keys may have different purposes.

For example:

- On D0 and D1, with the field named “memory usage on routing engine,” keys RE0 and RE1 represent two routing engines per device.
- There’s no guarantee that RE0 is a primary on all devices, therefore they might not be comparable when checking for outliers.
- This mechanism allows for selecting only the primaries: D0-RE0 and D1-RE1.

Algorithm

The outlier detection algorithms include k-fold, 3-sigma, and dbscan:

K-Fold Cross-Validation Using 3-Sigma—K-fold cross-validation using the 3-sigma (k-fold 3-sigma) algorithm is used to create a machine learning model for identifying outliers. K-fold cross-validation splits the entire data set into k groups and uses the following general process to create the machine learning models:

- Each unique k group is used as a test data set.
- The k groups (that are not being used as a test data set) are used as the training data set to build a machine learning model.
- Each test data set is evaluated using its associated machine learning model.
- The test data set with the most outliers relative to their associated machine learning model is classified as an outlier.

For example, if k is the number of devices in a device group and the group has 4 devices, then $k=4$. For cross-validation, four machine learning models are built and the test data sets are evaluated as follows:

- Model 1: Trained with the data points collected from device 1, device 2, and device 3, then tested with the data points collected from device 4.
- Model 2: Trained with the data points collected from device 1, device 2, and device 4, then tested with the data points collected from device 3.
- Model 3: Trained with the data points collected from device 1, device 3, and device 4, then tested with the data points collected from device 2.
- Model 4: Trained with the data points collected from device 2, device 3, and device 4, then tested with the data points collected from device 1.

Using the k-fold 3-sigma algorithm is more applicable if it's known that outliers will skew in one direction or another. If there are outliers on both sides of normal data points, or there are enough outlier data points to make the algorithm believe that nothing is outlying, then the k-fold 3-sigma algorithm will not provide significant results.

DBSCAN—Density-Based Spatial Clustering of Applications with Noise (DBSCAN) is an unsupervised machine learning algorithm used to create a machine learning model for identifying time series data sets as outliers:

- Time series data sets are grouped in such a way that data points in the same cluster are more similar to each other than those in other clusters.
- Clusters are dense regions of time series data sets, separated by regions of lower density.
- If a time series data set belongs to a cluster, it should be near many other time series data sets in that cluster.
- Time series data sets in lower density regions are classified as outliers.

Using the DBSCAN algorithm is more applicable if outliers appear inside the 3-sigma threshold of the other data points. DBSCAN can find outlying behavior that doesn't appear as a significant deviation from the normal behavior at any given time step.

Sigma coefficient (k-fold-3sigma only)

The sigma coefficient is a thresholding argument (default value is 3). The thresholding argument determines, at each point in time for a series, how far away a value must be from the other values to be marked as an outlier.

Sensitivity

Sensitivity is used to calculate the outliers, m , that the algorithm seeks to find in the data. Sensitivity determines the number of time series test data sets to return as outliers (the top m are returned):

- Sensitivity “low”: 0.03% of the number of sensors
- Sensitivity “medium”: 5% of the number of sensors
- Sensitivity “high”: 36% of the number of sensors
- Absolute percentage x : x *number of sensors (float, 0.0-1.0)

Learning period

See the [“Learning period” on page 58](#) description of the “Understanding HealthBot Anomaly Detection” section.

Understanding HealthBot Predict

IN THIS SECTION

- [Field | 63](#)
- [Algorithm | 63](#)
- [Learning period | 63](#)
- [Pattern periodicity | 63](#)
- [Prediction offset | 63](#)

This section describes the input parameters associated with HealthBot rules used for forecasting future values with the HealthBot median prediction machine learning algorithm or the Holt-Winters prediction machine learning algorithm. Once the machine learning models are built, they can be used in production

to predict trends and forecast future values. The accuracy of the results increases with larger amounts of data.

Field

See the [“Field” on page 57](#) description of the “Understanding HealthBot Anomaly Detection” section.

Algorithm

The HealthBot Predict feature uses either the median prediction algorithm, or the Holt-Winters prediction algorithm.

The median value represents the midpoint for a range of values within a data sampling. For every pattern periodicity bucket, a median is calculated from the data samples available in the bucket.

Learning period

See the [“Learning period” on page 58](#) description of the “Understanding HealthBot Anomaly Detection” section.

Pattern periodicity

See the [“Pattern periodicity” on page 59](#) description of the “Understanding HealthBot Anomaly Detection” section. For the median prediction algorithm, we recommend a minimum number of 10 data points for building a machine learning model. For the Holt-Winters algorithm, the pattern periodicity should be half or less of the learning period.

Prediction offset

The prediction offset value is a time in the future at which you want to predict a field value. For example, if the present time is 6th Feb 2019 10:00 and the prediction offset is set to 5 hours, then HealthBot will predict a field value for 6th Feb 2019 15:00.

Supported units of time for prediction offset include: seconds, minutes, hours, days, weeks, and years. You must enter the plural form of the time unit with no space between the number and the unit. For example, a prediction offset of one hour must be entered as 1hours.

HealthBot Rule Examples

IN THIS SECTION

- [HealthBot Anomaly Detection Example | 64](#)
- [HealthBot Outlier Detection Example | 73](#)

The machine learning HealthBot rules described in this section are available for upload from the [HealthBot Rules and Playbooks](#) GitHub repository.

HealthBot Anomaly Detection Example

IN THIS SECTION

- [Sensors \(check-icmp-statistics\) | 64](#)
- [Fields \(check-icmp-statistics\) | 65](#)
- [Variables \(check-icmp-statistics\) | 67](#)
- [Functions \(check-icmp-statistics\) | 68](#)
- [Triggers \(check-icmp-statistics\) | 68](#)
- [Rule Properties \(check-icmp-statistics\) | 73](#)

This example describes how the **check-icmp-statistics** Healthbot device rule is configured to send ICMP probes to user-defined destination hosts to detect anomalies when round trip average response time is above static or dynamic thresholds.

The following sections show how to configure the applicable input parameters for each HealthBot rule definition block (such as, Fields, Variables, and Triggers) using the HealthBot GUI. For more information about how to configure HealthBot rules, see [Creating a New Rule using the HealthBot GUI](#).

Sensors (check-icmp-statistics)

[Figure 9 on page 65](#) shows the general properties and iAgent sensor configured for the **check-icmp-statistics** rule. For information about the **count-var** and **host-var** variables, see [“Variables \(check-icmp-statistics\)” on page 67](#).

Figure 9: General properties (check-icmp-statistics) and Sensors definition (icmp)

The screenshot shows the configuration page for a rule named 'check-icmp-statistics'. At the top, there are buttons for 'Delete', 'Save', and 'Save & Deploy'. Below this, the description is 'Sends ICMP probes to destination host and notify anomalies' and the synopsis is 'ICMP response analyzer'. The 'Sensors' tab is active, showing a list of sensors with 'icmp' selected. The configuration for the 'icmp' sensor is as follows:

- Sensor Name:** icmp
- Sensor type:** iAgent
- File:** icmp_statistics.yml
- Table:** pingTable
- Frequency:** 60s
- Arg count:** {{count-var}}
- Arg host:** {{host-var}}

Fields (check-icmp-statistics)

The following fields are configured for the **check-icmp-statistics** rule:

dt-response-time—(See [Figure 10 on page 66](#)) Configuration for anomaly detection using the k-means algorithm. When an anomaly is detected, HealthBot returns a value of 1.

rtt-average-ms—(See [Figure 11 on page 66](#)) Round trip average response time.

rtt-threshold—(See [Figure 12 on page 67](#)) Static threshold for the round trip average response time. The **rtt-threshold** variable populates this field.

Figure 10: Fields definition (dt-response-time)

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

+ ADD FIELD

dt-response-time host packet-loss-percent rtt rtt-average-ms rtt-threshold

DELETE DT-RESPONSE-TIME

Field Name * dt-response-time

Description Dynamic threshold for field rtt-average-ms

Field Type integer

Add to Rule Key

Ingest type (Field source) Formula

Formula Anomaly Detection **Field *** \$rtt-average-ms **Algorithm *** k-means **Learning period *** 7d

Pattern periodicity * 1h

Figure 11: Fields definition (rtt-average-ms)

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

+ Add field

dt-response-time rtt-average-ms rtt-threshold

DELETE rtt-average-ms

Field name * rtt-average-ms

Description Round trip time

Field type float

Add to rule key

Ingest type (Field source) Formula

Formula micro-milli **Argument micros *** \$rtt-average

5008206

Figure 12: Fields definition (rtt-threshold)

The screenshot shows the 'Fields' configuration interface. On the left, a sidebar lists 'dt-response-time', 'rtt-average-ms', and 'rtt-threshold'. The main area is for editing 'rtt-threshold'. It includes a 'Field name' field with 'rtt-threshold', a 'Description' field with 'RTT response time static threshold', a 'Field type' dropdown set to 'string', an 'Add to rule key' toggle (unchecked), an 'Ingest type (Field source)' dropdown set to 'Constant', and a 'Constant value' field containing the variable reference '{{rtt-threshold-var}}'. A 'Delete rtt-threshold' button is in the top right.

Variables (check-icmp-statistics)

The following three variables are configured for the **check-icmp-statistics** rule:

count-var—(See [Figure 13 on page 67](#)) ICMP ping count. Count is set to 1 by default.

host-var—(See [Figure 14 on page 68](#)) Destination IP address or host name where ICMP probes are periodically sent.

rtt-threshold-var—(See [Figure 15 on page 68](#)) Static threshold value for round trip average response time. Threshold value is 1 ms by default. This variable populates the **rtt-threshold** field.

Figure 13: Variables definition (count-var)

The screenshot shows the 'Variables' configuration interface. On the left, a sidebar lists 'count-var', 'host-var', and 'rtt-threshold-var'. The main area is for editing 'count-var'. It includes a 'Variable name' field with 'count-var', a 'Default value' field with '1', a 'Type' dropdown set to 'Sensor Argument', and a 'Description' field with 'Input ICMP probe count'. A 'Delete count-var' button is in the top right.

Figure 14: Variables definition (host-var)

The screenshot shows the 'Variables' tab in a configuration tool. On the left, a list of variables includes 'count-var', 'host-var', and 'rtt-threshold-var'. The 'host-var' variable is selected and its details are shown in the main area. The 'Variable name' is 'host-var', the 'Default value' is 'Enter default value', and the 'Type' is 'Sensor Argument'. The 'Description' is 'Input Destination host'. A 'Delete host-var' button is in the top right.

Figure 15: Variables definition (rtt-threshold-var)

The screenshot shows the 'Variables' tab in a configuration tool. On the left, a list of variables includes 'count-var', 'host-var', and 'rtt-threshold-var'. The 'rtt-threshold-var' variable is selected and its details are shown in the main area. The 'Variable name' is 'rtt-threshold-var', the 'Default value' is '1', and the 'Type' is 'Integer'. The 'Description' is 'Input RTT response static threshold in milli seconds'. A 'Delete rtt-threshold-var' button is in the top right.

Functions (check-icmp-statistics)

Figure 16 on page 68 shows the function configured for the **check-icmp-statistics** rule. This function converts the unit of measure for the round trip average response time from microseconds to milliseconds.

Figure 16: Functions definition (micro-milli)

The screenshot shows the 'Functions' tab in a configuration tool. On the left, a list of functions includes 'micro-milli'. The 'micro-milli' function is selected and its details are shown in the main area. The 'Function name' is 'micro-milli', the 'Path to function' is 'micro_milli.py', and the 'Method name' is 'rtt_micro_milli'. The 'Description' is 'This function converts microseconds to milliseconds'. Under 'Arguments', there is one argument named 'micros' which is mandatory. An 'Add Argument' button is at the bottom left. A 'Delete micro-milli' button is in the top right.

Triggers (check-icmp-statistics)

The following triggers and terms are configured for the **check-icmp-statistics** rule:

- packet-loss — (See Figure 17 on page 69)

The following terms are configured for the packet-loss trigger:

is-device-not-reachable—(See [Figure 18 on page 70](#)) When the ICMP packet loss is 100%, the HealthBot health status severity level is set to major (red).

is-device-up—(See [Figure 19 on page 70](#)) When the packet loss is greater than 0, the severity level is set to minor (yellow).

no-packet-loss—(See [Figure 20 on page 71](#)) Otherwise, the severity level is set to normal (green).

- round-trip-time — (See [Figure 21 on page 71](#))

The following terms are configured for the round-trip-time trigger:

is-rtt-fine—(See [Figure 22 on page 72](#)) When the host is not reachable or the round trip average response time is above the static threshold, the HealthBot health status severity level is set to major (red).

is-rtt-medium—(See [Figure 23 on page 72](#)) When an anomaly is detected using the anomaly detection formula, HealthBot returns a value of 1 for the dt-response-time field, and the severity level is set to minor (yellow). In this case, the response time is above the anomaly detection.

rtt-normal—(See [Figure 24 on page 73](#)) Otherwise, the severity level is set to normal (green).

Figure 17: Triggers definition (packet-loss)

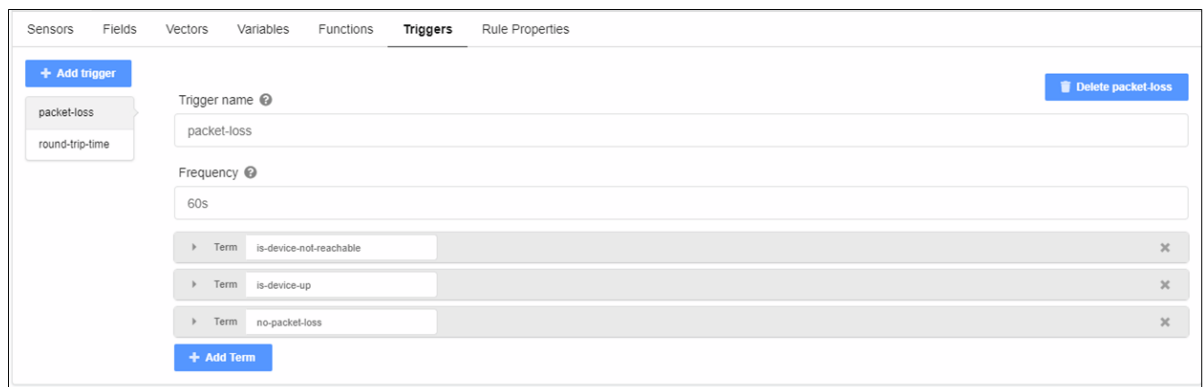


Figure 18: Terms definition (is-device-not-reachable)

The screenshot shows a configuration window for the term 'is-device-not-reachable'. It is divided into two main sections: 'WHEN' and 'THEN'.
In the 'WHEN' section, there are four fields: 'Left operand' set to '\$packet-loss', 'Operator' set to '==', 'Right operand' set to '100', and 'Time range' set to '2m'. A red minus sign is visible to the right of the time range field. Below these fields is a blue button labeled '+ Add Condition'.
In the 'THEN' section, there is a 'Color' dropdown menu with a red color swatch selected. Below that is a 'Message' text area containing the text 'Host \$host not reachable'. At the bottom of the 'THEN' section, there is a toggle switch for 'Evaluate next term' which is currently turned off, and a blue button labeled '+ Add Function'.
A vertical ID 's008213' is located on the right side of the window.

Figure 19: Terms definition (is-device-up)

The screenshot shows a configuration window for the term 'is-device-up'. It is divided into two main sections: 'WHEN' and 'THEN'.
In the 'WHEN' section, there are four fields: 'Left operand' set to '\$packet-loss', 'Operator' set to '>', 'Right operand' set to '0', and 'Time range' set to '2m'. A red minus sign is visible to the right of the time range field. Below these fields is a blue button labeled '+ Add Condition'.
In the 'THEN' section, there is a 'Color' dropdown menu with a yellow color swatch selected. Below that is a 'Message' text area containing the text 'Host \$host reachable and \$packet-loss % packet loss'. At the bottom of the 'THEN' section, there is a toggle switch for 'Evaluate next term' which is currently turned off, and a blue button labeled '+ Add Function'.
A vertical ID 's008214' is located on the right side of the window.

Figure 20: Terms definition (no-packet-loss)

The screenshot shows a configuration window for a term named 'no-packet-loss'. It is divided into two main sections: 'WHEN' and 'THEN'.
- The 'WHEN' section contains a single button labeled '+ Add Condition'.
- The 'THEN' section includes:
 - A 'Color' dropdown menu currently set to a green color.
 - A 'Message' text area containing the text: 'Host \$host reachable and \$packet-loss % packet loss'.
 - A toggle switch for 'Evaluate next term', which is currently turned off.
 - A button labeled '+ Add Function' at the bottom.

Figure 21: Triggers definition (round-trip-time)

The screenshot shows the 'Triggers' configuration page in a web interface. The top navigation bar includes 'Sensors', 'Fields', 'Vectors', 'Variables', 'Functions', 'Triggers', and 'Rule Properties'.
- On the left, there is a sidebar with a '+ Add trigger' button and a list of triggers: 'packet-loss' and 'round-trip-time'.
- The main area is for configuring the 'round-trip-time' trigger, with a 'Delete round-trip-time' button in the top right.
- Fields for configuration include:
 - 'Trigger name' (with a help icon) set to 'round-trip-time'.
 - 'Frequency' (with a help icon) set to '60s'.
 - A list of terms, each with a right-pointing arrow, a text input, and a delete 'x' icon:
 - Term: 'is-rt-fine'
 - Term: 'is-rt-medium'
 - Term: 'rt-normal'
 - An '+ Add Term' button at the bottom.

Figure 22: Terms definition (is-rtt-fine)

Term: is-rtt-fine

WHEN

Left operand	Operator	Right operand	Time range
\$rtt-average-ms	>=	\$rtt-threshold	Enter a time range

+ Add Condition

THEN

Color: ■

Message: Round trip time(\$rtt-average-ms ms) to \$host is above static threshold

Evaluate next term

+ Add Function

5008217

Figure 23: Terms definition (is-rtt-medium)

Term: is-rtt-medium

WHEN

Left operand	Operator	Right operand	Time range
\$dt-response-time	==	1	Enter a time range

+ Add Condition

THEN

Color: ■

Message: Round trip time(\$rtt-average-ms ms) to \$host is above dynamic threshold

Evaluate next term

+ Add Function

5008218

Figure 24: Terms definition (rtt-normal)

The screenshot shows a configuration window for a term named 'rtt-normal'. It is divided into two main sections: 'WHEN' and 'THEN'.
 - The 'WHEN' section contains a single button labeled '+ Add Condition'.
 - The 'THEN' section contains:
 - A 'Color' dropdown menu currently set to a green color swatch.
 - A 'Message' text area containing the text: 'Round trip time(\$rtt-average-ms ms) to \$host is normal'.
 - A toggle switch labeled 'Evaluate next term' which is currently turned off.
 - A button labeled '+ Add Function' at the bottom.

Rule Properties (check-icmp-statistics)

Figure 25 on page 73 shows the rule properties configured for the **check-icmp-statistics** rule.

Figure 25: Rule Properties definition (check-icmp-statistics)

The screenshot shows the 'Rule Properties' configuration page for the 'check-icmp-statistics' rule. The page has several tabs: Sensors, Fields, Vectors, Variables, Functions, Triggers, and Rule Properties (which is selected).
 The configuration includes the following fields:
 - **version**: 1
 - **Contributor**: juniper
 - **author-email**: Enter author-email
 - **Date**: (empty)
 - **supported-healthbot-version**: 2.0.0
 - **Supported Device**: Juniper Devices
 - **Juniper Devices**: Junos
 - **Product Name**: MX
 - **Release Name**: 11.4
 - **Release Support**: min-supported-releases
 - **Platform**: All

HealthBot Outlier Detection Example

IN THIS SECTION

- Sensors (check-outlier) | 74
- Fields (check-outlier) | 74
- Variables (check-outlier) | 75
- Triggers (check-outlier) | 75
- Rule Properties (check-outlier) | 77

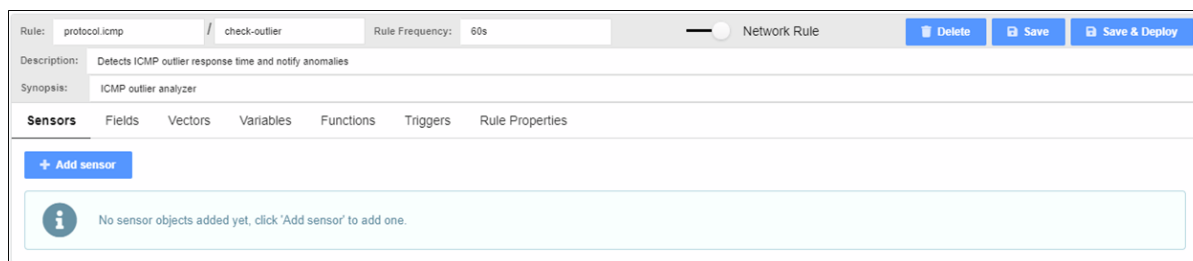
This example describes how the **check-outlier** Healthbot network rule is configured to detect outliers across the devices in a device group using the round trip time average response time.

The following sections show how to configure the applicable input parameters for each HealthBot rule definition block (such as, Fields, Variables, and Triggers) using the HealthBot GUI. For more information about how to configure a HealthBot rule, see [Creating a New Rule using the HealthBot GUI](#).

Sensors (*check-outlier*)

[Figure 26 on page 74](#) shows the general properties configured for the **check-outlier** rule. Note that this rule is a network rule.

Figure 26: General properties (check-outlier)



Fields (*check-outlier*)

[Figure 27 on page 75](#) shows the field configured for the **check-outlier** rule. This field defines the DBSCAN algorithm and **rtt-xpath** variable for outlier detection. For information about the **rtt-xpath** variable, see [“Variables \(check-outlier\)” on page 75](#).

The results of the HealthBot outlier detection algorithm are stored in a table in the times series database. Each row in the table contains outlier detection output and metadata that is associated with a particular time series. You can use the information in the table to configure HealthBot rule triggers. Each column in the table is identified by a unique name that starts with the user-defined outlier detection field name. For example, you can use the *field-name-is-outlier* (**rtt-ol-is-outlier**) and *field-name-device* (**rtt-ol-device**) column names to configure a trigger that detects outliers and produces a message that indicates which specific device was determined to be the outlier (see [“Triggers \(check-outlier\)” on page 75](#)).

Figure 27: Fields definition (rtt-ol)

The screenshot shows the 'Fields' configuration page for a field named 'rtt-ol'. The page has tabs for 'Sensors', 'Fields', 'Vectors', 'Variables', 'Functions', 'Triggers', and 'Rule Properties'. The 'Fields' tab is active. On the left, there is a list of fields with 'rtt-ol' selected. On the right, the configuration form includes:

- Field name***: rtt-ol
- Description**: Outlier detection of devices in a group using xpath
- Field type**: string
- Add to rule key**:
- Ingest type (Field source)**: Formula
- Formula**: Outlier Detection
- Dataset***: {{rtt-xpath}}
- Algorithm***: dbscan
- Sensitivity***: low
- Learning Period***: 30m

Buttons for '+ Add field' and 'Delete rtt-ol' are visible. A vertical ID '5006222' is on the right edge.

Variables (check-outlier)

Figure 28 on page 75 shows the variable configured for the **check-outlier** rule. This variable defines the devices in the network from which HealthBot collects round trip average response time data for the outlier detection machine learning models.

Figure 28: Variables definition (rtt-xpath)

The screenshot shows the 'Variables' configuration page for a variable named 'rtt-xpath'. The page has tabs for 'Sensors', 'Fields', 'Vectors', 'Variables', 'Functions', 'Triggers', and 'Rule Properties'. The 'Variables' tab is active. On the left, there is a list of variables with 'rtt-xpath' selected. On the right, the configuration form includes:

- Variable name***: rtt-xpath
- Default value**: *
- Type***: String
- Description**: Input xpath of devices to be in outlier detection format e.g. /device-group[device-group-name=core]/device[device-id=R0]/topic[topic-name=protocol.icmp]/rule[rule-name=check-icmp-statistics]/rtt-average-ms,/device-group[device-group-name=core]/device[device-id=R1]/topic[topic-name=protocol.icmp]/rule[rule-name=check-icmp-statistics]/rtt-average-ms,/device-group[device-group-name=core]/device[device-id=R2]/topic[topic-name=protocol.icmp]/rule[rule-name=check-icmp-statistics]/rtt-average-ms

Buttons for '+ Add variable' and 'Delete rtt-xpath' are visible. A vertical ID '5006223' is on the right edge.

Triggers (check-outlier)

Figure 29 on page 76 shows the trigger configured for the **check-outlier** rule. The following terms are configured for the **icmp-outlier-detection** trigger:

is-outlier-detected—(see Figure 30 on page 76) When an outlier is detected, HealthBot returns a value of 1 for the **rtt-ol-is-outlier** field, and the HealthBot health status severity level is set to major (red). This term also produces a message that indicates which specific device was determined to be the outlier.

no-outlier—(See Figure 31 on page 77) Otherwise, HealthBot returns a value of 0, and the severity level is set to normal (green).

Figure 29: Triggers definition (icmp-outlier-detection)

The screenshot shows the 'Triggers' tab in a configuration tool. At the top, there are tabs for 'Sensors', 'Fields', 'Vectors', 'Variables', 'Functions', 'Triggers', and 'Rule Properties'. Below the tabs, there is a '+ Add trigger' button on the left and a 'Delete icmp-outlier-detection' button on the right. The main configuration area includes a 'Trigger name' field with the value 'icmp-outlier-detection', a 'Frequency' field with the value '60s', and a list of terms. The list contains two terms: 'is-outlier-detected' and 'no-outlier', each with a right-pointing arrow and a close button (X). At the bottom of the list is an '+ Add Term' button. A vertical ID 's008224' is visible on the right side of the interface.

Figure 30: Terms definition (is-outlier-detected)

The screenshot shows the 'Terms' configuration window for the term 'is-outlier-detected'. The window has a title bar with 'Term is-outlier-detected' and a close button (X). The main area is divided into 'WHEN' and 'THEN' sections. The 'WHEN' section contains a table with four columns: 'Left operand', 'Operator', 'Right operand', and 'Time range'. The 'Left operand' is '\$rtt-ol-is-outlier', the 'Operator' is '==', and the 'Right operand' is '1'. The 'Time range' field is empty with the placeholder text 'Enter a time range' and a red minus sign. Below the table is an '+ Add Condition' button. The 'THEN' section includes a 'Color' dropdown menu set to red, a 'Message' text area containing 'Outlier detected on \$rtt-ol-device', and an 'Evaluate next term' toggle switch which is currently turned off. At the bottom, there is an information icon and a message: 'Functions can be used as Trigger actions too, define them using the 'Functions' menu at the top.' A vertical ID 's008225' is visible on the right side of the interface.

Figure 31: Terms definition (no-outlier)

Term: no-outlier

WHEN

Left operand: \$rtt-ol-is-outlier | Operator: == | Right operand: 0 | Time range: Enter a time range

+ Add Condition

THEN

Color: Green

Message: No outlier detected on \$rtt-ol-device

Evaluate next term

Functions can be used as Trigger actions too, define them using the 'Functions' menu at the top.

Rule Properties (check-outlier)

Figure 32 on page 77 shows the rule properties configured for the **check-outlier** rule.

Figure 32: Rule Properties definition (check-outlier)

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

version: 1 | Contributor: juniper | author-email: Enter author-email | Date: | supported-healthbot-version: 2.0.0

Supported Device

Juniper Devices

Junos

Product Name: MX

Release Name: 11.4 | Release Support: min-supported-releases | Platform: All

Release History Table

Release	Description
3.1.0	Starting with HealthBot Release 3.1.0, you can choose the Holt-Winters prediction algorithm

Frequency Profiles and Offset Time

IN THIS SECTION

- [Frequency Profiles | 78](#)
- [Offset Time Unit | 83](#)

Frequency Profiles

Frequency profiles are a central location in which sensor and rule time frequencies can be managed. To understand frequency profiles, consider the following.

When defining rules in HealthBot you can:

- Define multiple rules that use the same sensor
- Define different sensor frequencies for each of the rules
- Apply all of these rules to the same device group/devices

This creates complexity in rule application and frequency adjustments within the individual rules:

- A key, consisting of sensor-path for OpenConfig and Native GPB sensors, or the tuple of file and table for iAgent sensors is used to identify the specific rules.
- HealthBot takes the minimum defined frequency for that sensor from the applied rules and uses it to subscribe to, or fetch, data from the devices.
- This make it hard to identify what the data rate should be for that sensor. To do that, you would have to go through the all the applied rules.
- A change in the sensor frequency of an applied rule might not take effect as intended.

To address these complexities, HealthBot needed a common place from which to control these frequencies.

Starting in HealthBot 3.0.0, frequency profiles can be created that allow you to manage sensor and rule frequencies from a single location and then apply the profiles in various locations in HealthBot. Application of these profiles allows for persistent and repeatable behavior in regard to frequencies for rules, sensors, triggers, formulas, references, learning periods, and hold times.

A sensor profile consists of a profile name and two optional sections: the sensors section and the non-sensors section. In each section, an entry consists of a sensor or rule name and a frequency. Frequency profiles are applied to device groups or network groups.

The steps for configuration are shown below.

Configuration Using HealthBot GUI

Frequency profiles are configured and managed in the HealthBot GUI or in the HealthBot CLI. In the GUI, they are managed by navigating to the **Settings > Ingest Settings** page and selecting the **Frequency Profile** tab from the left side of the page.

NOTE: While the sections of the frequency profile are both optional, at least one section must be filled out per frequency profile if you want the applied profile to be able to do anything.

Add a Frequency Profile

1. Click the **+ FREQUENCY PROFILE** button

The Add Frequency Profile window appears.

2. Give the profile a name such as **Profile1**

3. (Optional) Click the **+ ADD SENSORS** button.

4. In the **Sensor Name** field, enter the sensor name as per the following guidelines:

- *OpenConfig Sensors:* Enter the OpenConfig path for the desired sensor, such as **/components** or **/interfaces**.
- *iAgent Sensors:* Enter the table name used in the sensor definition, such as **ChassisAlarmTable** or **REutilizationTable**
- *SNMP:* Enter the sensor name such as **npr_qmon_ext**
- *BYOI:* Enter **<topic-name/rule-name/sensor-name>**, such as **topic1/rule1/sensor1**

5. In the **Frequency** field, enter the appropriate frequency, such as **30seconds**, **1minute**, **2hours**, and so on.

6. (Optional) Click the **+ ADD NON-SENSORS** button.

7. In the **Rule Name** field, enter the rule name such as **check-chassis-alarms**.

8. In the **Frequency** field, enter the appropriate frequency, such as **45seconds**, **3minutes**, **1hour**, and so on.

Repeat steps 3 through 5 or 6 through 8 as desired for the profile.

An example result of the previous steps might look like:

Figure 33: Edit a Frequency Profile

Edit Frequency Profile: Profile1

Name
Profile1

SENSORS

Sensor Name	Frequency *	
/components	60seconds	-
/interface	30seconds	-

+ ADD SENSORS

NON-SENSORS

Rule Name	Frequency *	
net-topic/net-rule	2minutes	-

+ ADD NON-SENSORS

CANCEL SAVE SAVE & DEPLOY

9. Click the **SAVE & DEPLOY** button to save and deploy the profile.

The new sensor profile is added to the list.

To edit an existing frequency profile:

1. Click the *<Profile Name>* from the list.
2. Make the needed changes as shown in the previous section.
3. Click the **SAVE** button to save the profile for later deployment or the **SAVE & DEPLOY** button to save and deploy immediately.

To delete an existing frequency profile:

1. Click the **Trash Can** icon to the right of the profile name.

2. Click the **DELETE** button to delete the profile but not deploy the change or the **DELETE & DEPLOY** button to delete and deploy immediately.

Usage Notes:

- *Profile Entries*—Multiple entries can be configured in each section as shown in [Figure 33 on page 80](#).
- *Override*—Sensor or rule frequency defined within an applied frequency profile overrides those defined within the individual rule or sensor.
- *Order of Precedence*—If a sensor or rule is defined in multiple frequency profiles, each with different frequency settings, the minimum frequency value for the sensor or rule is used.

Configuration Using HealthBot CLI

In the HealthBot CLI, you can configure the same frequency profile described above. An example of the CLI configuration needed to complete the example above looks like:

```
user@mgd-69ab987fbc6-pt9sh> show configuration healthbot ingest-settings
frequency-profile Profile1
```

```
sensor /components {
    frequency 60seconds;
}
sensor /interface {
    frequency 30seconds;
}
non-sensor net-topic/net-rule {
    frequency 2minutes;
}
```

Apply a Frequency Profile Using the HealthBot GUI

Frequency profiles are applied to HealthBot device groups or network groups. When you create or edit a device or network group, you apply frequency profiles by selecting them from the **Ingest Frequency** section of the **Device Group** definition. [Figure 34 on page 82](#) below shows an example of frequency profiles being applied to a device group.

Figure 34: Apply Frequency Profiles

Edit "ix-lab-group"

Description

Describe the device group.

Devices*

spice ✕

Select devices to add to this group.

Timezone **Retention Policy**

+/-00:00 ▼

Specify the Timezone in the format +/-hh:mm. Select the name of the retention policy to be applied.

<p>Native Ports</p> <p>Comma separated values</p> <p>Specify the native sensors receiver port(s).</p>	<p>Flow Ports</p> <p>Comma separated values</p> <p>Specify the netflow sensor receiver port(s).</p>	<p>Syslog Ports</p> <p>Comma separated values</p> <p>Specify the syslog messages receiver UDP port(s).</p>
--	--	---

Flow Deploy Nodes

Select flow ingest deploy nodes for this device group.

Reports

Select reports to generate for this device group.

Summarization ▼

Ingest Frequency ▲

Ingest Frequency Profiles

Profile1 ✕ Profile2 ✕ ✕ ▲

All profiles...

NewProfile

Notifications ▼

Once you have applied the needed profiles, save and deploy the device group using the **SAVE & DEPLOY** button.

BEST PRACTICE: It is strongly recommended that you only apply frequency profiles to rules that make use of the [“Offset Time Unit” on page 83](#) feature.

Apply a Frequency Profile Using the HealthBot CLI

An example of a device group CLI configuration which includes a frequency profile and could be deployed in HealthBot is shown below.

```
user@mgd-69ab987fbc6-pt9sh> show configuration healthbot device-group lab-group
```

```
devices router1;
ingest-frequency Profile1;
```

Offset Time Unit

The HealthBot offset time unit is used in conjunction with the [“Frequency Profiles” on page 78](#) to automatically manage time range calculations in various places within HealthBot rules. To understand the HealthBot offset function, consider the following scenario:

In HealthBot, you can define a rule which

- Uses a sensor to gather data with a frequency of 10 seconds
- Has a field that calculates the mean every 60 seconds

If you later decide to increase the frequency of the sensor to 60 seconds, then calculating the mean every 60 seconds would not make any sense. The result is that you would have to manually update the field calculation any time up want to change the sensor frequency.

Starting with HealthBot Release 3.0.0, you can set the time range for the mean calculation to a value of 60, or 60offset, rather than 60s, or 60 seconds. Using an offset time value rather than a static time value tells HealthBot to automatically multiply the sensor frequency by the numeric value of the offset. Thus, any change to sensor frequency will automatically be included in the time range calculation for a formula.

An offset time unit can be used in place of standard time units in the following time range locations:

- Formulas
- References
- Vectors
- Trigger Frequency
- Trigger Term

Below we discuss GUI and CLI examples of how to configure the offset time unit in each of the locations mentioned above.

Offset Used in Formulas

In this example, we are creating a rule, *Rule1*, with a sensor, *Sensor1*. The sensor frequency is set to 10 seconds.

In [Figure 35 on page 84](#) below, you can see the **Fields** block definition for *Rule1* with the **Sensors** block definition framed in green. The formula, *formula1*, has a **Time range** value of 20

Figure 35: Offset in a Formula

The screenshot displays the configuration page for a rule named 'Rule1'. The 'Fields' tab is selected, showing a field named 'formula1' with a 'Time range' of '20'. An inset window shows the 'Sensors' tab for 'Sensor1' with a 'Frequency' of '10s'. The 'Time range' field in the main window is highlighted with a green border.

A CLI formatted configuration snippet for the same field looks like:

```
user@mgd-97bb5d555-sw87n$ show healthbot topic external rule rule1
synopsis "This rule is used only to demonstrate various rule features and concepts";
description "Demonstration Rule";
sensor Sensor1 {
  open-config {
    sensor-name /interfaces;
    frequency 10s;
  }
}
field field1 {
  constant {
    value 833;
  }
}
```

```

    }
    type integer;
}
field formulal {
  formula {
    max {
      field-name "$field1";
      time-range 20;
    }
  }
  type integer;
}
}

```

Usage Notes for Offset Used in Formulas

- An offset value applied to the time range of a formula multiplies the rule, sensor, or trigger frequency of that rule.
- The result of this example is that the time range for the Max formula is now 2 times the *Sensor1* frequency of 10 seconds, or 20 seconds ($2 * 10s = 20s$).
- If a frequency profile of 30 seconds is applied to the rule used in the example, then the resulting time-range would be 60 seconds ($2 * 30s = 60s$).
- Offset time can be applied to the following formulas: latest, count, min, max, sum, mean, on-change, and stdev.

Offset Used in References

In this example, there are two rules in play, but only one is shown. The unseen rule, Rule1, is in the topic routing-engines and is named routing-engines (routing-engines/routing-engines). Rule1 has a frequency of 20 seconds. Rule1 is referenced .

Rule2, shown in [Figure 36 on page 86](#), is a network rule which has a reference field named *ref1*. The field, *ref1*, references back to Rule1 through the **Reference XPath Expression** with a **Time Range** setting of 3offset.

Figure 36: Offset Time Used in Reference Field

The screenshot shows the configuration page for a Network Rule. At the top, the rule is named 'Rule2' with a frequency of '10s'. The description is 'Demonstration Rule' and the synopsis states it is used for demonstration purposes. The 'Fields' tab is active, showing a list of fields on the left with 'ref1' selected. The configuration for 'ref1' includes:

- Field Name:** ref1
- Description:** Add a description for this field
- Field Type:** integer
- Add to Rule Key:** (disabled)
- Ingest type (Field source):** Reference
- Reference XPath expression:** `"/device-group[device-group-name='Core4']/device[device-id='R1']/topic[topic-name='routing']`
- Time Range:** 3offset
- Data if missing:** Default value

Usage Notes for Offset Used in References

- Offset values used in references multiply the frequency of the referenced rule by the offset value. In this case, the 20 second frequency of Rule1 is multiplied by 3, resulting in a 60 second time-range ($3 * 20s = 60s$).
- If a frequency profile of 60 seconds (60s) was applied to Rule1, the time range for the reference would increase to 180 seconds ($3 * 60s = 180s$).

Offset Used in Vectors

In this example, there are 3 rules at play. Rules *Rule1* and *Rule2* are not shown but are referenced by the vector.

Rule1 is in topic line-cards and is named line-cards (line-cards/line-cards) and has a frequency of 20 seconds (20s).

Rule2 is in topic routing-engines and is named routing-engines (routing-engines/routing-engines) and has a frequency of 30 seconds (30s).

In *Rule3* below, we have defined a vector, *vector1* that consists of 2 path references and 1 field. The vector has a **Time Range** defined as 3offset. [Figure 37 on page 87](#) shows the vector block definition in the HealthBot GUI.

Figure 37: Offset Time Used in Vector Block

The screenshot shows the HealthBot GUI configuration for a rule named 'Rule3'. The rule is of type 'external' and has a frequency of '10s'. The 'Vectors' tab is active, showing a single vector named 'vector1'. The vector configuration includes:

- Vector Name:** vector1
- Ingest Type:** path
- References:**
 - "/device-group[device-group-name='Core4']/device[device-id='R1']/topic[topic-name='routing-engines']/rule[rule-name='routing-engines']/field[slot='0']/ref1"
 - "/device-group[device-group-name='Core4']/device[device-id='R1']/topic[topic-name='line-cards']/rule[rule-name='line-cards']/memory"
 - \$formula1
- Time-Range:** 3offset

Usage Notes for Offset Used in Vectors

- An offset value defined in the time range of a vector multiplies the sensor or rule frequency of the referenced rule or sensor. If a field from the same rule is used as the reference, then the frequency of the rule containing the vector is used.
- When multiple references or fields are defined for a vector and offset time is used, the offset value is applied to each path independently. So, for this example in which our offset value is 3 (3offset):
 - The path reference to Rule1: `"/device-group[device-group-name='Core4']/device[device-id='R1']/topic[topic-name='line-cards']/rule[rule-name='line-cards']/memory"` which has a frequency of 20 seconds, would result in a time range of 60 seconds for the vector ($3 * 20s = 60s$).
 - The path reference to Rule2: `"/device-group[device-group-name='Core4']/device[device-id='R1']/topic[topic-name='routing-engines']/rule[rule-name='routing-engines']/field[slot='0']/ref1"` which has a frequency of 30 seconds, would result in a time range of 90 seconds for the vector ($3 * 30s = 90s$).

NOTE: There are no spaces or line breaks in the path references. They are added to this document only to enhance readability.

- The field reference to **formula1**: Since **formula1** is a normal field used within the vector block, the rule frequency of the current rule is used as a basis. So, the time range for **formula1** is 30 seconds ($3 * 10s = 30s$).
- If a frequency profile of 60 seconds is applied to Rule1 (line-cards/line-cards), then the time range for that path in the vector would be 180 seconds ($3 * 60s = 180s$).
- If a frequency profile of 120 seconds is applied to Rule2 (routing-engines/routing-engines), then the time range for that path in the vector would be 360 seconds ($3 * 120s = 360s$).
- The time range for the field reference, **formula1**, would remain the same as before at 30 seconds unless a change is made to Rule3 or a frequency profile with a different time is applied to Rule3.

Offset Used in Triggers

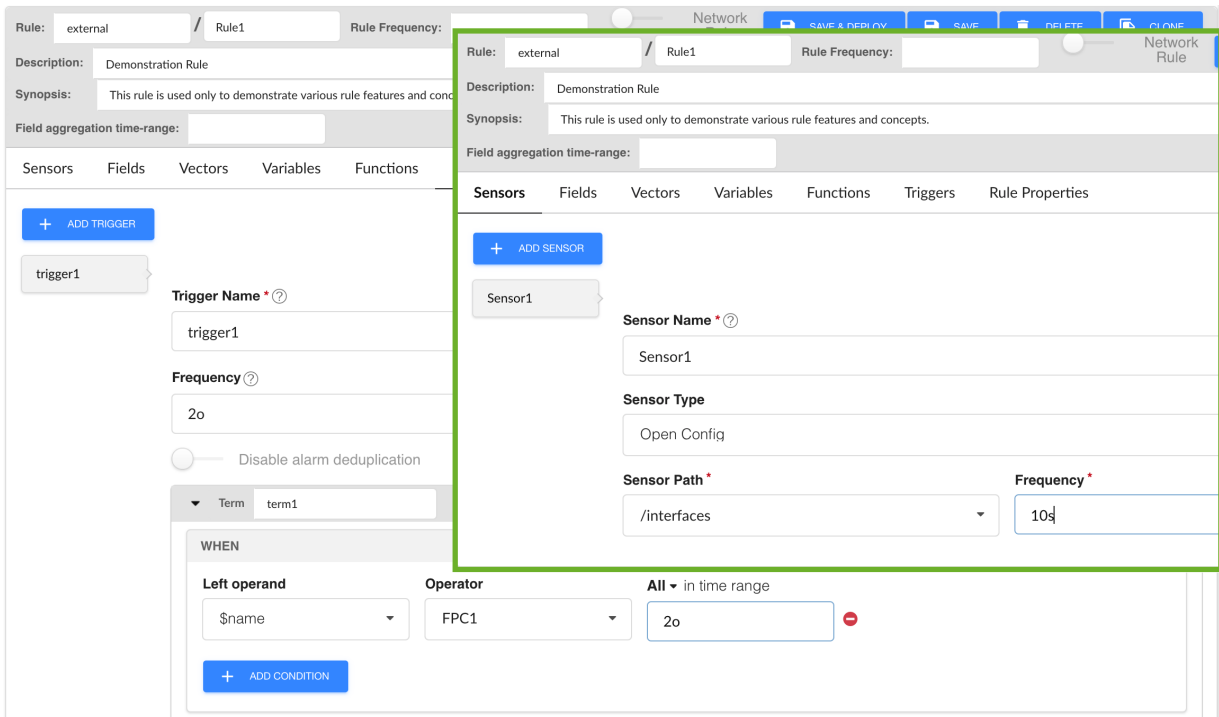
In this example, we have one rule, Rule1. [Figure 38 on page 88](#) below shows the trigger block definition with the sensor block definition overlaid with a green border.

The rule, *Rule1*, has a sensor frequency of 10 seconds (10s) applied.

The trigger itself, *Trigger1*, has an offset frequency of 2 (2o) applied.

The trigger term, *term1*, has its own time range offset of 2 (2o) applied as well.

Figure 38: Offset Time Used in Trigger Block



Usage Notes for Offset Time Used in Triggers

- An offset value defined in trigger frequency multiplies the sensor or rule frequency. So, the offset value of 2 in *trigger1* causes the trigger frequency to be interpreted as 20 seconds ($2 * 10s = 20s$) because the sensor frequency of the rule is used as the basis.
- An offset value defined in a trigger term multiplies the trigger frequency value. So, the offset value of 2 in *term1* causes the term frequency to be interpreted as 40 seconds ($2 * 20s = 40s$).

Offset Used in Trigger Reference

In this example, we have 2 rules in the topic external, and one frequency profile, *prof1*.

The rules are:

- *external/test*: The *test* rule has a sensor named *components* which is an OpenConfig sensor with a sensor path of */components* and a sensor frequency of 10 seconds.

It also has a trigger, *trig1* with a frequency of 2o or 2offset. The trigger has a term named *Term_1*, which is not used in the example.

Rule: external / test

Description: Description

Synopsis: Synopsis

Field aggregation time-range:

Sensors Fields Vectors Variables

+ ADD SENSOR

components

Sensor Name *

components

Sensor Type

Open Config

Sensor Path *

/components

Frequency *

10s

Sensors Fields Vectors Variables Functions Triggers

+ ADD TRIGGER

trig1

Trigger Name * ?

trig1

Frequency ?

2o

Disable alarm deduplication

Term Term_1

+ ADD TERM

- *external/ref*: The *ref* rule is a non-sensor rule, which means that the rule uses a reference field, *trigger_reference*, to reference the sensor defined in another rule; in this case *external/test*.

Rule: / Rule Frequency: Network Rule

Description:

Synopsis:

Field aggregation time-range:

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

Field Name *

Description

Field Type

Add to Rule Key

Ingest type (Field source)

Reference XPATH expression * **Time Range**

And the frequency profile is:

- *prof1*: The *prof1* profile sets the frequency for the */components* sensor at 30 seconds.

Edit Frequency Profile: prof1

Name

SENSORS

Sensor Name	Frequency *
<input type="text" value="/components"/>	<input type="text" value="30seconds"/> ⊖

[+ ADD SENSORS](#)

NON-SENSORS

[+ ADD NON-SENSORS](#)

[CANCEL](#) [SAVE](#) [SAVE & DEPLOY](#)

HealthBot Licensing

IN THIS SECTION

- [HealthBot Licensing Overview | 91](#)
- [Managing HealthBot Licenses | 94](#)

HealthBot Licensing Overview

Juniper Networks introduced the Juniper Flex Software Subscription Licensing model to provide an efficient way for customers and partners to manage licenses for hardware and software features. Contrail HealthBot uses this licensing model.

To use a licensed feature, you need to purchase and activate a license that corresponds to that feature and deploy that license so that it can be utilized by the software during normal operation. You can administer

and manage the licenses through the [Juniper Agile Licensing Portal](#). The portal provides an intuitive, task-based user interface that provides full lifecycle management of licenses.

HealthBot supports the standalone mode for deploying licenses. Standalone mode allows you to activate a license on a software instance. Such a license can only be used by the instance on which it is activated. Sharing a license with multiple instances is not permissible.

HealthBot has three service levels:

HealthBot Standard (Free)—This is the free model solution and is available for customers that have a valid support contract. No license is required.

HealthBot Advanced—This is the first tier of the fee-based model solution and requires that you purchase a HealthBot Advanced feature license. You may purchase additional device-based feature licenses depending on your business needs.

HealthBot Premium—This is the second tier of the fee-based model solution and requires that you purchase a HealthBot Premium feature license. You may purchase additional device-based feature licenses depending on your business needs.

The following table shows a comparison between the HealthBot service models:

HealthBot Standard (Free)	HealthBot Advanced (Fee-based)	HealthBot Premium (Fee-based)
Free with a valid Juniper support contract.	Access to advanced features. You must purchase a HealthBot Advanced feature license. Additional device feature licenses available.	Access to premium features. You must purchase a HealthBot Premium feature license. Additional device feature licenses available. The premium feature set currently includes Time Series Database (TSDB) High Availability (HA).
No license required.	Subscription licenses.	Subscription licenses.
Limit of 5 custom playbooks and 10 custom rules.	Unlimited custom playbooks and custom rules.	Unlimited custom playbooks and custom rules.
Number of devices allowed is based on your contractual agreement.	Number of devices allowed is based on the number of device feature licenses purchased.	Number of devices allowed is based on device feature licenses purchased.
—	Full support for generating reports.	Full support for generating reports.
—	Full support for generating notifications.	Full support for generating notifications.

HealthBot Standard (Free)	HealthBot Advanced (Fee-based)	HealthBot Premium (Fee-based)
–	Limited support for Time Series Database (TSDB) HA settings. You cannot configure TSDB HA settings.	Full support for changing TSDB HA and other TSDB settings.
–	Support for default and custom rules with machine learning features such as dynamic thresholds, outlier detection, median prediction, and microburst.	Support for default and custom rules with machine learning features such as dynamic thresholds, outlier detection, median prediction, and microburst.
–	Support for default and custom rules with user-defined functions.	Support for default and custom rules with user-defined functions.
–	Multivendor telemetry data collection.	Multivendor telemetry data collection.

*For information about HealthBot product options or to obtain a trial license, contact your local sales representative.

The following table lists the available HealthBot device feature licenses:

Device Feature Name	Description
HBOT-C1	Extra small devices such as small remote CPEs, small branch SRX, and small ACX.
HBOT-C2	Small switches such as fixed form factor EX, ACX, and QFX.
HBOT-C3	Small chassis-based switches, routers, and firewalls such as MX10003 and PTX10K3
HBOT-C4	Large chassis-based systems like the MX10K, PTX, and large SRX.

NOTE: In HealthBot releases prior to 3.1.0, the feature licenses were called HBOT-BASE, HBOT-G1, and HBOT-G2. License purchases made in previous versions will be honored in HealthBot 3.1.0. New license purchases as of 3.1.0 can not be applied to previous versions of HealthBot. See your local account manager for details.

For the HealthBot 3.1.0 release, enforcement of device license counts is based on the total number of all C1-C4 licenses installed minus the total number of licenses actually used. This enforcement model may change in future releases of HealthBot.

Managing HealthBot Licenses

IN THIS SECTION

- [Add a License to HealthBot | 94](#)
- [View Licensing Status in HealthBot | 94](#)

Once you have obtained a HealthBot license through the [Juniper Agile Licensing Portal](#), you can:

Add a License to HealthBot

To add a HealthBot license:

1. Click on the **Settings > License Management** option in the left-nav bar.
2. Click the **+ License** button.
3. Click the **Choose File** button in the pop-up window.
4. Navigate to the license file you want to add, and then click **Open**.
5. Click the **Add** button to add the license file.

The file should now appear in the **Licenses Added** table on the License Management page. For information about the **Licenses Added** table, see [“Licenses Added” on page 95](#).

View Licensing Status in HealthBot

IN THIS SECTION

- [Features Summary | 95](#)
- [Licenses Added | 95](#)

The License Management page consists of two tables; the Features Summary table and the Licenses added table. Details of the table contents are shown below.

Features Summary

The following table describes the HealthBot feature license attributes in the Features Summary table:

Attribute	Description
Feature	HealthBot license name. HBOT-Premium —Limit and usage count for premium feature licenses for this instance of HealthBot HBOT-Advanced —Limit and usage count for advanced feature licenses for this instance of HealthBot HBOT-Devices —Maximum number of devices that can be managed by this instance of HealthBot based on the device feature licenses added.
Description	Brief description of the HealthBot feature license.
License Limit	The number of valid HealthBot licenses successfully added and available for use.
Usage Count	The number of available licenses that are currently in use in this instance of HealthBot.
Valid Until	Date and time when the license expires.
Compliance	Color definitions for dot indicator: Green —Feature licenses are in compliance with Juniper's End User License Agreement. Yellow —Device feature licenses are $\geq 90\%$ of the limit. You are getting close to running out of licenses. This status is only applicable to device feature licenses. Red —Feature licenses are not in compliance with Juniper's End User License Agreement. Click on the red dot to view details about the compliance issue.

Licenses Added

The following table describes the HealthBot license attributes in the **Licenses Added** table. Click the caret next to the **License ID** to view the features that are provided by the license ID.

Attribute	Description
License ID	Identification number for the HealthBot license generated through the Juniper Agile Licensing Portal.

Attribute	Description
SKU Name	Name of the HealthBot software licensing package.
Customer ID	Identification name for the customer.
Order Type	Types include: Commercial, demo, education, emergency, lab, and unknown.
Validity Type	Types include: Date-based or permanent.
Start Date	Start date of the HealthBot license.
End Date	End date of the HealthBot license.
Feature ID	Identification number for the feature license.
Feature Name	HealthBot feature license name. For more information, see “Features Summary” on page 95.
Feature Description	Brief description of the HealthBot feature license.
License Count	<p>HBOT-Premium—Allow premium capabilities in HealthBot</p> <p>HBOT-Advanced—Allow advanced capabilities in HealthBot</p> <p>HBOT-C1—Number of C1 devices that can be managed by HealthBot..</p> <p>HBOT-C2—Number of C2 devices that can be managed by HealthBot.</p> <p>HBOT-C3—Number of C3 devices that can be managed by HealthBot.</p> <p>HBOT-C4—Number of C4 devices that can be managed by HealthBot .</p>

Release History Table

Release	Description
3.1.0	For the HealthBot 3.1.0 release, enforcement of device license counts is based on the total number of all C1-C4 licenses installed minus the total number of licenses actually used.

2

CHAPTER

Management and Monitoring

Manage HealthBot Users and Groups | **98**

Manage Devices, Device Groups, and Network Groups | **104**

HealthBot Rules and Playbooks | **118**

Monitor Device and Network Health | **148**

Alarms and Notifications | **181**

Generate Reports | **200**

Configure a Secure Data Connection for HealthBot Devices | **216**

Configure Data Summarization | **219**

Modify the UDA and UDF Engines | **222**

Logs for HealthBot Services | **227**

Troubleshooting | **230**

HealthBot Configuration - Backup and Restore | **240**

Manage HealthBot Users and Groups

IN THIS SECTION

- [User Management | 98](#)
- [Group Management | 99](#)
- [Limitations | 104](#)

Starting with Release 3.0.0, HealthBot employs role-based access control (RBAC) to control access to the user interface and HealthBot tools and objects. RBAC is applied to user groups that are made up of a list of users.

The use of access controls within HealthBot allows you to grant one group of users, like operators, read-only access to certain pages like **Configuration > Device Configuration**; while granting a different group of users, like administrators, read-write access to that same page.

Default User and First Login

When HealthBot is first installed, the default username and password are set as *admin* and *healthbot* respectively. The admin user has complete control over all of HealthBot's access controls. The credentials above are used for the first login at the URL **https://<HealthBot hostname or IP>:8080**.

Upon successful first login and before the admin user is granted access to the GUI, they are required to create a new password. The **Temporary Password Reset** window pops up and provides instructions regarding password length, capitalization, special characters, and so on. Once you save this password, a pop-up window notifies you that the password has been changed. From this time forward, the admin user logs in with the new password.

Once the admin user is logged in, all user and group management is carried out on the **Administration > User Management** page.

User Management

The **User Management** page is the first page shown when you navigate to **Administration > User Management** from the left-nav bar. This page is used to:

- **View a list of current HealthBot users**

The list shows user details including first name, last name and status. User status can be active (green) or inactive (red).

- **Add new users**

Click the + to bring up the **Add User(s)** window.

- **Edit existing users**

Select an existing user by clicking anywhere on that user's line in the list. Then click the **Edit User (Pencil)** icon to bring up the **Edit <username>** window. You can change any parameter except the username.

- **Export the user list**

With no user selected, click the **Export (Page with arrow)** icon to bring up the export dialog.

- **Delete a user**

Select an existing user by clicking anywhere on that user's line in the list. Then click the **Delete User (Trash Can)** icon. Confirm the action and the user is deleted.

NOTE:

- There is currently no self-service type of lost password mechanism. Password reset must be done manually by an administrator with read-write access to the **User Management** page. The administrator must edit the user, change the password, and then notify the user by appropriate means.
- If you set a user's status to inactive or delete that user, they are immediately prevented from logging in to HealthBot through the login page.

Group Management

A user group is a collection of roles to which a HealthBot user can be assigned. The roles within a user group define the access (read-only or read-write) that all members of the group have in common. In other words, user groups are where RBAC controls are applied.

The **User Groups** page is accessed by navigating to **Administration > User Management** from the left-nav and selecting **User Groups** on the left side of the **User Management** page.

- **View a list of current HealthBot user groups**

The list shows user group details including group name and description.

- **Add new user groups**

Click the + to bring up the **Add Group** window.

Starting in HealthBot Release 3.1.0, RBAC has been enhanced to include the roles selector helper. The roles selector helper appears when you add or edit a user group. See [Figure 39 on page 101](#).

Figure 39: Add User Group

Add Group

Group Name*
uGroup1

Unique Name for the group

Group Description

New user group for documentation purposes.

Description of the group

ROLES SELECTOR HELPER

Search ×

Functionality	Read	Write
> Dashboard	<input checked="" type="checkbox"/>	<input type="checkbox"/>
> Monitor	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
> Configuration	<input checked="" type="checkbox"/>	<input type="checkbox"/>
> Settings	<input type="checkbox"/>	<input type="checkbox"/>
> Administration	<input type="checkbox"/>	<input type="checkbox"/>

Include Access Roles

SELECT ALL

CLEAR

Selected Roles

System Roles*

- configuration
- configuration
- data-store
- data-store
- data
- debug
- device-groups
- device
- devices
- event
- files
- first-login
- health-tree
- health
- license
- login
- logout
- mgd
- network-groups
- playbook
- playbooks
- sensors
- system-settings
- token
- topic
- topics
- tsdb-counters
- user-profile
- user-profile

GUI Roles

- ui-configuration-device
- ui-configuration-devicegroups
- ui-configuration-network
- ui-configuration-playbooks
- ui-configuration-rules
- ui-dashboard-device-list
- ui-dashboard-device-status
- ui-dashboard-device-vendors
- ui-dashboard-devicegroup-list
- ui-dashboard-devicegroup-status
- ui-dashboard-network-status
- ui-dashboard-networkgroup-list
- ui-dashboard-tsdb
- ui-monitor-alarms
- ui-monitor-device-health
- ui-monitor-devicegroup-health
- ui-monitor-graph
- ui-monitor-network-health

Associated Users

CANCEL

SAVE

SAVE & DEPLOY

- **Edit existing user groups**

Select an existing user group by clicking anywhere on that group's line in the list. Then click the **Edit User (Pencil)** icon to bring up the **Edit <groupname>** window.

NOTE: When you add or edit a user group, the window has sections called **System Roles** and **GUI Roles** under the **Selected Roles** pull-down. These sections show the specific read-only (R) or read-write (W) permissions that are assigned to the group as a result of the selections made in the **ROLES SELECTOR HELPER**.

- **Export the user group list**

With no user group selected, click the **Export (Page with arrow)** icon to bring up the export dialog.

- **Delete a user group**

Select an existing user group by clicking anywhere on that group's line in the list. Then click the **Delete User (Trash Can)** icon. A confirmation window appears. Confirm the action (Save and Deploy) to complete the deletion. The pre-defined user groups *hbdefault* and *hbadmin* cannot be deleted.



WARNING: Adding and editing user groups in HealthBot is an advanced feature that requires a deep understanding of the available roles and how they apply to RBAC. We recommend that you use only the Role Selector check-boxes to add or remove permissions. We do not recommend that you add or remove individual system or GUI roles.

Pre-Defined User Groups

HealthBot is shipped with four pre-defined user groups:

- *hboperator*—Provides login capability and the ability to manage your own profile. Each user belongs to this group.
- *hbmonitor*—Provides read-only access to read and observe any configured entity in Healthbot.
- *hbconfig*: Provide all the capabilities of the *hbmonitor* group plus the ability to modify any configuration in HealthBot.
- *hbadmin*—Provide all the capabilities of the *hbconfig* group plus the ability to manage users and groups.

None of the pre-defined user groups can be changed or removed. The default *admin* user is automatically a member of the *hbadmin* group. The default admin user is the first member of this group and cannot be removed from it. Additional administrator users can be added to this group by the admin user or another member of the *hbadmin* group.

Limitations

In HealthBot Release 3.1.0, the RBAC implementation is limited in some ways:

- The available roles, such as **R-Devices**, **W-Devices**, **R-Datastore**, etc. are all pre-defined. There is no way to add new roles or delete existing roles.
- All roles are endpoint driven, not specific to any resource. This means that if you have read permission for devices, you can read all devices in the system. There is no means to restrict the read access to a subset of devices.
- Roles are permissive in nature. You cannot create a role that blocks access to any given endpoint such as rules. If a user is created but not given any group membership, they will not be able to access the HealthBot GUI.
- RBAC is currently limited to API service. This means that if you have read-only access to a page such as **Configuration > Devices**, you can see the entire page and interact with all of its controls. You could even go through the motions of creating a device in the GUI. However, when you click **SAVE** or **SAVE & DEPLOY** an API is called and it will recognize that you do not have the required permission to create a device. Errors are displayed at that time.
- If you migrate data from your existing 2.1.X installation to your 3.0.0 or later installation, user data is not migrated. Any existing users must be recreated manually, by the admin user, after migration.

Release History Table

Release	Description
3.1.0	Starting in HealthBot Release 3.1.0, RBAC has been enhanced to include the roles selector helper
3.0.0	Starting with Release 3.0.0, HealthBot employs role-based access control (RBAC) to control access to the user interface and HealthBot tools and objects.

Manage Devices, Device Groups, and Network Groups

IN THIS SECTION

- [Adding a Device | 106](#)
- [Editing a Device | 109](#)

- [Adding a Device Group | 109](#)
- [Editing a Device Group | 113](#)
- [Configuring a Retention Policy for the Time Series Database | 113](#)
- [Adding a Network Group | 114](#)
- [Editing a Network Group | 117](#)

Use the appropriate Configuration pages from the left-nav to manage devices, device groups, and network groups. HealthBot supports both Junos devices by default and third party vendor devices with the required license installed. You must add a device to one or more device groups or create a network group before you can apply HealthBot rules and playbooks to a device. Network groups allow you to correlate health

status data between multiple devices across the network. For example, you can create a network group that monitors the ping times between two or more devices and notifies you if the ping times are too high.

Adding a Device

To add a device:

1. Click the **Configuration > Device** option in the left-nav 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	Unique system identifier required for JTI native sensors. Junos devices use the following format: <code><host_name>:<jti_ip_address></code> 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.
Flow Source IPs	Enter the IP address(es) that this device uses to send NetFlow data to HealthBot.
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.
Vendor	Vendor or supplier of the device you are using. NOTE: If you plan to use Cisco IOS XR devices, you must first configure the telemetry. For more information, see <i>HealthBot Installation Requirements</i>
OS	Device operating system.
SNMP Port Number	Port number required for SNMP.
SNMP Community	Community name required for SNMP. Default is public.
Timezone	Timezone for this device, specified as + or -hh:mm. For example, +07:00
Syslog Source IPs	List of IP addresses for the device sending syslog messages to HealthBot. For example, 10.10.10.23, 192.168.10.100.

Attributes	Description
Syslog Hostnames	List of hostnames for the device sending syslog messages to HealthBot. For example, router1.example.com.
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>
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-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 Devices table, see [“Monitor Device and Network Health” on page 148](#).

Editing a Device

To edit a device:

1. Click the **Configuration > Device** option in the left-nav bar.
2. Click anywhere on the line that contains the device name in the table under **DEVICES**.
You can search and filter the device names in the table.
3. Click the **Pencil** (Edit Device) icon.
4. Modify the attributes, as needed.
See [“Adding a Device” on page 106](#) for a description of each attribute.
5. 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” on page 148](#).
6. (Optional) A device can be deleted by clicking the **Trash Can** (Delete Device) icon with the device selected.

Adding a Device Group

To add a device group:

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	Add devices to the device group from the drop-down list. (Required)
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 HealthBot. The port numbers must be unique across the entire HealthBot installation.
Syslog Ports	Specify the UDP port(s) on which syslog messages are received by HealthBot.
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. For information on how to configure a retention policy, see “Configuring a Retention Policy for the Time Series Database” on page 113 .
Reports	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). 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 . For more information, see “Alarms and Notifications” on page 181 .

Attributes	Description
Summarization	<p>To improve the performance and disk space utilization of the HealthBot time series database, you can configure data summarization methods to summarize the raw data collected by HealthBot. 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 “Configure Data Summarization” on page 219.</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>
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 HealthBot devices. To receive HealthBot 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 “Alarms and Notifications” on page 181.</p>

Attributes	Description
Logging Configuration	<p>You can collect different severity levels of logs for the running HealthBot services of a device group. 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 HealthBot 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 “Logs for HealthBot Services” on page 227.</p>
Publish	<p>You can configure HealthBot to publish HealthBot 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 HealthBot 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” on page 148](#).

Editing a Device Group

To edit a device group:

1. Click the **Configuration > Device Group** option in the left-nav bar.
2. Click on the device group name under **DEVICE GROUPS**.
3. Click on the **Pencil** (Edit Device Group) icon.
4. Modify the attributes, as needed.
See [“Adding a Device Group” on page 109](#) for a description of each attribute.
5. 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” on page 148](#).
6. (Optional) A device group can be deleted by clicking the **Trash Can** (Delete Device Group) icon with the device group selected.

Configuring a Retention Policy for the Time Series Database

To configure a retention policy for the time series data used for root cause analysis (RCA):

1. Click the **Settings > System** option in the left-nav bar.
2. Select **Retention Policy Settings**.
3. Click the **+ Retention Policy** button.
4. Enter the necessary values in the text boxes for the retention policy.

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

Attributes	Description
Name	Name of the retention policy.
Duration	Amount of time the root cause analysis (RCA) data is retained in the HealthBot RCA database. By default, data is retained for 7 days. The data must be entered in hours or days. For example, 1 day is entered as 1d or 24h.

5. Click **Save** to save the configuration or click **Save and Deploy** to save and deploy the configuration.

You can now apply the retention policy to a device group. For information on how to apply a retention policy to a device group, see [“Adding a Device Group” on page 109](#).

Adding a Network Group

To add a network group:

1. Click the **Configuration > Network** option in the left-nav bar.
2. Click the + (Add Network) button.
3. Enter the necessary values in the text boxes and select the appropriate options for the network group.

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

Attributes	Description
Name	Name of the network group. (Required)
Description	Description for the network group.
Reports	<p>In the Reports field, select one or more health report profile names from the drop-down list to generate reports for the network 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 “Alarms and Notifications” on page 181.</p>
Notifications	<ul style="list-style-type: none"> • You can use the Alarm Manager feature to organize, track, and manage KPI alarm notifications received from HealthBot devices. • To receive HealthBot 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 “Alarms and Notifications” on page 181.</p>
Ingest Frequency	Select existing Ingest Frequency Profiles to override rule or sensor frequency settings.

Attributes	Description
Logging Configuration	<p>You can collect different severity levels of logs for the running HealthBot services of a network group. 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 HealthBot service for the network 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 “Logs for HealthBot Services” on page 227.</p>
Publish	<p>You can configure HealthBot to publish HealthBot sensor and field data for a specific network 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 HealthBot rule topic and rule name pairs that contain the field data you want to publish.</p>

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 network, see [“Monitor Device and Network Health” on page 148](#).

Editing a Network Group

To edit a network group:

1. Click the **Configuration > Network** option in the left-nav bar.
2. Click anywhere on the line that contains the group name in the table under **NETWORK LIST**.
3. Click on the **Edit Network (Pencil)** icon.
4. Modify the attributes, as needed.
See [“Adding a Network Group” on page 114](#) for a description of each attribute.
5. Click **Save** to save the configuration or click **Save and Deploy** to save and deploy the configuration. For information on how to use the network group cards, see [“Monitor Device and Network Health” on page 148](#).
6. (Optional) A network can be deleted by clicking the **Delete Network (Trash Can)** icon.

RELATED DOCUMENTATION

[HealthBot Rules and Playbooks | 118](#)

[Monitor Device and Network Health | 148](#)

HealthBot Rules and Playbooks

IN THIS SECTION

- [Add a Pre-Defined Rule | 119](#)
- [Create a New Rule Using the HealthBot GUI | 119](#)
- [Edit a Rule | 136](#)
- [Add a Pre-Defined Playbook | 137](#)
- [Create a New Playbook Using the HealthBot GUI | 138](#)
- [Edit a Playbook | 139](#)
- [Manage Playbook Instances | 140](#)

The device or network performance elements that are important to one company may not be important to another. HealthBot uses Rules and Playbooks to define key performance indicators (KPIs) and organize them into groups that are applied to network devices.

This document presents the tasks involved in creating, editing, and deleting HealthBot rules and playbooks.

Add a Pre-Defined Rule

Juniper has created a set of pre-defined rules that you can use to gather information from various Juniper components and the networks they reside in. You can add these rules to HealthBot at any time. After installation, many default pre-defined rules appear in the Rules and Playbooks pages. Pre-defined rules cannot be changed or removed; however, you can clone any rule (pre-defined or user defined) simply by clicking the **CLONE** button on the upper-right part of the rule definition. A cloned rule goes to the *external* topic and can be re-configured at will.

To upload additional pre-defined rules to HealthBot:

1. Using a browser, go to <https://github.com/Juniper/healthbot-rules> and download the pre-defined rule file to your system.
2. In the HealthBot GUI, click the **Configuration > Rules** icon in the left-nav bar.
3. Click the **↑ Upload Rule Files** button.
4. Click the **Choose Files** button.
5. Navigate to the rule file and click **Open**.
6. Select one of the following options:

Upload	Upload the file and save the rule within the defined topic area 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.
Upload & Deploy	Upload the file, save the rule within the defined topic area, and immediately deploy the configuration.

Create a New Rule Using the HealthBot GUI

IN THIS SECTION

● Rule Filtering | 121

- Sensors | 123
- Fields | 125
- Vectors | 128
- Variables | 130
- Functions | 131
- Triggers | 133
- Rule Properties | 136

To create a new rule using the HealthBot GUI, you'll first fill out general descriptive information about the rule and then navigate through several rule definition blocks in the Rules page to provide the specific configuration for the HealthBot rule.

To start creating a new HealthBot rule:

1. Click the **Configuration > Rules** icon in the left-nav bar. A list of HealthBot rules organized by HealthBot topic is displayed along the left side of the Rules page.
2. Click the add rule button (+ **Add Rule**).
3. Enter general descriptive information about the rule using the following input parameters:

Parameter	Description
Rule	<p>For a new rule, this parameter is pre-populated with <i>external / user_rule_random characters</i>, for example, external / user_rule_2p0ghk. The fields separated by the slash (/) represent HealthBot topic name and HealthBot rule name, respectively.</p> <p><i>external</i> is the topic name used for user-defined topics. For the HealthBot rules pre-defined by Juniper, Juniper has curated a set of pre-defined device component-based topic names. For more information about HealthBot topics, see HealthBot Topics.</p> <p>Replace the user_rule_random characters rule name with a name that appropriately represents the rule's description such as packets-in, packets-out, system_memory, etc.</p>
Rule frequency	(Network rule only) Specify how often data for the network rule is collected by HealthBot. This setting is overridden if the rule is included in a frequency profile that is applied to a network group.

Parameter	Description
Description	(Optional) Enter a detailed description for the rule.
Synopsis	(Optional) Enter a brief description for the rule. The synopsis is displayed when you hover over the rule name listed along the left side of the Rules page.
Field Aggregation Time Range	This optional value defines how often HealthBot aggregates the data received by the sensor. This helps reduce the number of data point entries in the time series database.

- (Network rule only) If the new rule is a network rule, toggle the Network rule switch to the right.
- Configure the rule definition blocks as needed.

Located directly below the Synopsis input parameter, you'll find links to the following rule definition blocks: **Sensors**, **Fields**, **Vectors**, **Variables**, **Functions**, **Triggers**, and **Rule Properties**. The following sections describe the input parameters for each of these rule definition blocks.

- Select one of the following options to save the new rule:

Save	Save the rule within the defined topic area 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.
Save & Deploy	Immediately deploy the configuration and save the rule within the defined topic area.

The following sections describe the input parameters for each of the HealthBot rule definition blocks:

Rule Filtering

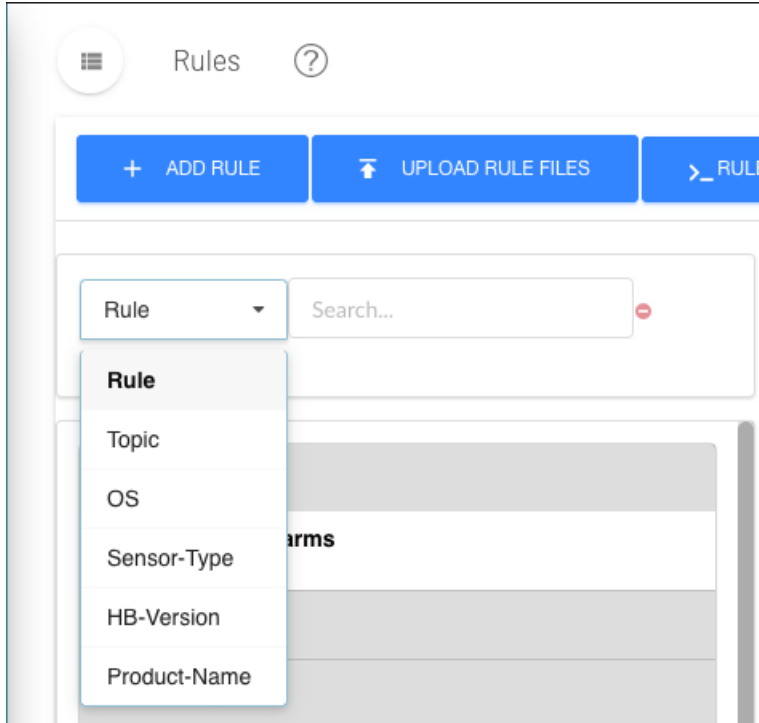
Starting in HealthBot Release 3.0.0, you can filter the Topics and Rules displayed on the left side of the Rules page. This allows you to quickly find rules that you are looking for. The search function works for topics, rules, sensor-types and other categories; working not only on titles, but also on the defined contents of rules.

The following procedure explains this filtering feature.

- Navigate to **Configuration > Rules** in the left-nav bar.

The **Rules** page is displayed. To the left of the rule definition area is a new section as shown in [Figure 40 on page 122](#) below.

Figure 40: Rule Filtering



2. From the pull-down menu, select the type of search you want to perform.
3. In the search field, begin entering your search text.

The topic list below shrinks to display only topics and rules that match your search criteria.

Sensors

Start configuring the new rule using the **Sensor** block. [Figure 41 on page 123](#) shows the sensor definition for the OpenConfig sensor **pppoe-error-statistics**.

Figure 41: A Sensor Definition

The screenshot displays the configuration page for a sensor. At the top, the rule is identified as 'service.protocols / pppoe-error-statistics' with a 'Rule Frequency' field. Below this, the 'Description' is 'Collects the PPPoE error periodically and notifies in case of anomalies' and the 'Synopsis' is 'Monitors PPPoE error statistics'. A navigation bar includes 'Sensors', 'Fields', 'Vectors', 'Variables', 'Functions', 'Triggers', and 'Rule Properties'. The 'Sensors' tab is selected, showing a '+ Add sensor' button and a list of sensors, with 'pppoe-error-statistics' highlighted. The configuration form contains the following fields:

- Sensor Name**: pppoe-error-statistics
- Sensor type**: Open Config
- Sensor path**: /junos/system/subscriber-management/client-protocols/ppp
- Frequency**: 60s

1. Click the add sensor button (+ **Add Sensor**).

A new sensor definition appears and is named **Sensor_random characters**, like **Sensor_2kgf04**.

2. Change the sensor name to something that makes sense for the rule you are defining.
3. From the drop-down list, choose the sensor type. You can choose one of: **OpenConfig**, **Native GPB**, **iAgent**, **SNMP**, **Syslog**, or **NetFlow**.

The required elements for defining the **Sensor Type** change depending on the selection you make. The frequency is expressed in #s, #m, #h, #d, #w, #y, or #o where # is a number and s, m, h, d, w, y specifies seconds, minutes, hours, days, weeks, years, and offset respectively. The o expression is used for defining an offset multiplier for use in formulas, references, triggers, learning periods, and hold-times.

The following list describes the elements that change based on your choice of **Sensor Type**. None of the other rule elements change because of a **Sensor Type** selection.

- **OpenConfig**—**Sensor path** is defined from a drop-down list of available OpenConfig sensors. **Frequency** refers to how often, in seconds, the sensor reports to HealthBot. The frequency can be overridden if the sensor is included in a frequency profile.
- **Native GPB**—**Sensor path** refers to the path for a Native GPB sensor. **Port** refers to the GPB port over which the sensor communicates with HealthBot.

- **iAgent—File** is the name of a YAML-formatted file that defines the NETCONF-accessible sensor. **Table** is defined from a drop-down list of available PyEZ tables and views in the YAML file. **Frequency** refers to how often the sensor is polled by HealthBot and can be overridden by including the sensor in a frequency profile.

Based on the table you've selected, input fields for a target or dynamic arguments might also be provided. For these additional fields, you can do one of the following:

- Leave the input field blank. No default value will be applied.
 - Enter a fixed value that will remain constant.
 - Enter a variable name enclosed in double curly/flower brackets (for example, `{{test-variable}}`)
The variable name must belong to a variable that was previously defined in the HealthBot rule, and the variable's **Type** option must be set to **Sensor Argument**.
- **SNMP—Table** is defined from a drop-down list of available SNMP tables. **Frequency** refers to how often, in seconds, HealthBot polls the device for data and can be overridden by including the sensor in a frequency profile.
 - **Syslog—Pattern set** is a user-configured element that includes one or more patterns (you configure a pattern for each event you want to monitor). The **Maximum hold period** is used in advanced cases and refers to the maximum time that the system will wait when correlating events using multiple patterns within a pattern set.

NOTE: The syslog sensor requires some pre-configuration. See *Syslog Ingest* for more details.

- **Flow—Template Name** is a Juniper-supplied built-in list of NetFlow v9 and IPFIX templates.

Fields

A sensor will generally carry information about multiple things. For example, a sensor that watches interfaces has information about all of the interfaces on the device. In HealthBot, we call these things Fields. Now that you've defined a sensor, you'll need to tell HealthBot which fields you're interested in.

1. Click the **Fields** link.

The screen updates and shows the defined field objects, if any, as shown in [Figure 42 on page 126](#)

Figure 42: The Fields Block

The screenshot displays the 'Fields' configuration page. At the top, there are navigation tabs: Sensors, **Fields**, Vectors, Variables, Functions, Triggers, and Rule Properties. On the left, a sidebar contains a '+ Add Field' button and a list of field names: 'cpu-util-percent', 'daemon', 'max-threshold', 'min-threshold', and 'process-name'. The main configuration area is for the 'cpu-util-percent' field. It has a 'Field name' field containing 'cpu-util-percent', a 'Description' field containing 'Daemon CPU utilization', and a 'Field type' dropdown menu with options: 'string', 'string', 'integer', and 'float'. Below this is a toggle switch for 'Add to rule key' which is currently turned off. The 'Ingest type (Field source)' dropdown is set to 'Sensor'. Underneath, the 'Sensor' dropdown is set to 'SystemProcExtTable' and the 'Path' field contains 'wcpu'. At the bottom, there is a 'Where (filter using expression)' section with a '+ Add Expression' button.

2. Click the add field button (+ **Add Field**).

3. Replace the random field name with a name that make sense for the rule you are defining, such as **interface-name**, **configured-threshold**, etc.
4. (Optional) Add descriptive text for the new field.
5. Set the appropriate **Field type**. The options for field type are: string, integer, and float. String is the default field type.
6. (Optional) Toggle the **Add to rule key** switch.

The add rule to key switch tells HealthBot that this field should be indexed and searchable. For example, when you enable this switch, the field name will be listed on the Devices page under the **Keys** column.

7. Select the appropriate ingest type (Field source) from the pull-down menu.

The following list shows the options available for the **Ingest type (Field source)** menu.

- **Sensor**–Use this or another sensor definition.
 - **Path**–Follow this Open Config or Netconf path within the sensor definition to gather specific data like the names of the interfaces. For iAgent sensors the **Path** refers to the path defined in the YAML file.
 - **Where**–Filter the available data to gather information about a specific element within, like a specific interface. This field can reference the **Variables** defined elsewhere within the rule. When referencing variables, use moustache notation, enclosed in slashes, such as: **{{interface_name}}**.
 - **Zero suppression**–For some sensors associated with devices running Junos OS, such as Junos Telemetry Interface Open Config and native GPB sensors, no field data is sent from the sensor when the data's value is zero. Enable the zero suppression switch to set the field data value to zero whenever no field data is sent from the sensor.
 - **Data if missing**–Specify a value as the default data value whenever no data is sent from the sensor. The format of the specified value should match the defined field type (string, integer, or float). If the zero suppression switch is also enabled, then the specified data-if-missing value is ignored, and the default sensor data value is set to zero.
- **Reference**–A reference to a field or trigger value from another rule.
 - **Data if missing**–Specify a value as the default data value whenever no reference data is fetched. The format of the specified value should match the defined field type (string, integer, or float).
- **Constant**–Use a constant when referring to a **Variable** defined within the rule, whose value doesn't change, such as **IO_Drops_Threshold**. A constant can also be a string or numeric value that does not change and is not a reference to a variable.

- **Constant value**–Use moustache notation to reference the variable like this: `{{IO_Drops_Threshold}}`.
 - **Formula**–Select the desired mathematical formula from the **Formula** pull-down menu.
8. (Optional) Set the **Field aggregation time-range**. Located above the Fields tab with the general rule parameters, this periodic aggregation setting helps to reduce the number of data points entered in the database. For example, if the sensor settings specify that HealthBot ingests data every 10 seconds, you could configure this setting to aggregate and record the relevant field data, say, every 60 seconds. Note that when using this setting, any field-specific time ranges must use the same value.

NOTE: Starting in HealthBot Release 3.1.0, you can add fields and keys to rules based on whether the incoming data meets user-defined conditions defined in tagging profiles. Tagging profiles are defined in HealthBot under **Settings > Ingest Settings** on the left-nav. See [“Healthbot Tagging” on page 36](#) for details.

Vectors

(Optional) Now that you have a sensor and fields defined for your rule, you can define vectors.

A vector is used when a single field has multiple values or when you receive a value from another field.

1. Click on the **Vectors** link. [Figure 43 on page 128](#) shows the Vectors block for a newly added vector.

Figure 43: Vectors Block

The screenshot displays the configuration page for a new vector. At the top, there are navigation tabs: Sensors, Fields, Vectors (selected), Variables, Functions, Triggers, and Rule Properties. A blue button with a plus sign and the text '+ Add vector' is located on the left. Below this, a grey box contains the vector name 'vector_sd19e3'. The main configuration area includes:

- Vector name***: A text input field containing 'vector_sd19e3'.
- Ingest Type**: A dropdown menu with 'path' selected.
- List of fields**: A dropdown menu with 'subscriber-count-maximum' selected. Other options visible are 'subscriber-count-minimum', 'total-subscriber-count', and 'total-subscribers-pred'.
- Time-range**: A text input field containing '7d'.

2. Click the add vector button (+ **Add Vector**)

3. Replace the random vector name with a name that makes sense for your rule.
4. Select an ingest type from the drop-down list. The additional input fields will vary depending on the selection you make.

For path:

Parameter	Description
List of fields	Select a field from the drop-down list. The list of fields is derived from all of the defined fields in this rule.
Time-range	Specify a time range from which the data should be collected. The time range is expressed in #s, #m, #h, #d, #w, #y where # is a number and s, m, h, d, w, y specifies seconds, minutes, hours, days, weeks, and years respectively. For example, enter 7 days as 7d.

For formula:

Parameter	Description
Formula Type	Select a formula type from the drop-down list: unique —Creates a vector with unique elements from another vector. and —Compares two vectors and returns a vector with elements common to both vectors. or —Compares two vectors and returns a vector with elements from both vectors. unless —Compares two vectors and returns a vector with elements from the left vector but not the right vector.
Vector name	(Unique formula type only) Select a vector name from the drop-down list. The list of vectors is derived from all of the defined vectors in this rule.
Left vector	Select a vector name from the drop-down list. The list of vectors is derived from all of the defined vectors in this rule.
Right vector	Select a vector name from the drop-down list. The list of vectors is derived from all of the defined vectors in this rule.

Variables

(Optional) The **Variables** block is where you define the parts of the sensor that you are interested in. For example, a rule that monitors interface throughput needs to have a way to identify specific interfaces from the list of available interfaces on a device. [Figure 44 on page 130](#) shows the **Variables** block for the `chassis.power/check-pem-power-usage` rule. The field details are discussed below.

Figure 44: The Variables Block

The screenshot shows the 'Variables' configuration page. At the top, there are tabs for 'Sensors', 'Fields', 'Vectors', 'Variables', 'Functions', 'Triggers', and 'Rule Properties'. The 'Variables' tab is active. On the left, there is a '+ Add Variable' button and a list of existing variables, including 'pem_power_usage_threshold'. The main form has three input fields: 'Variable name' with the value 'pem_power_usage_threshold', 'Default value' with the value '888888880', and a 'Description' field containing the text 'Enter PEM power usage threshold value'. To the right, a 'Type' dropdown menu is open, displaying a list of options: Integer, Floating Point, String, Boolean, Device, and Device Group. The 'Integer' option is currently selected. A 'Delete pem_power_usage_threshold' button is located at the top right of the form area.

1. Click on the **Variables** tab.
2. Click the add variable button (+ **Add Variable**)
3. Replace the random **Variable name** with a variable name that makes sense for your rule, such as **pem-power-usage-threshold**.

BEST PRACTICE: The accepted convention within Juniper for naming of elements within HealthBot is to always start with a lower-case letter and use hyphens to separate words. Make sure that your variable names are unique, clearly named, and follow a recognizable pattern so that others can understand what the variable is for. Any abbreviations should be used consistently throughout.

4. Set an appropriate default value in the **Default value** field.

Default values vary depending on field type. Integer field types use numeric default values, while string field types use strings to set exact defaults and regular expressions that allow you to set the default from a list. Any default values set during rule definition can be overridden at apply-time at either the device or device group level.

5. Select the appropriate variable type from the **Type** pull-down menu.

Available field types are: Integer, Floating Point, String, Boolean, Device, and Device Group.

Functions

(Optional) Define any needed functions.

The **Functions** block allows users to create functions in a python file and reference the methods that are available in that file. The python file must be created outside of HealthBot. You must know about the method names and any arguments because you will need those when defining the function.

[Figure 45 on page 131](#) shows the **Functions** block for the `chassis.power/check=pem-power-usage` rule. The field details are discussed below.

Figure 45: The Functions Block

The screenshot shows the 'Functions' configuration page in HealthBot. At the top, there are navigation tabs: Sensors, Fields, Vectors, Variables, **Functions**, Triggers, and Rule Properties. Below the tabs, there is a blue button labeled '+ Add Function'. To the left of the main form, there is a list of existing functions, with 'used-percentage' highlighted. The main form contains the following fields:

- Function name ***: A text input field containing 'used-percentage'.
- Path to function ***: A dropdown menu showing 'used-percentage.py'.
- Method name ***: A text input field containing 'used_percentage'.
- Description**: A text area containing 'calculates % of power usage out of total available'.
- Arguments**: A section with two arguments:
 - Name**: 'total', with a 'Mandatory' toggle switch that is turned on.
 - Name**: 'used', with a 'Mandatory' toggle switch that is turned on.

At the bottom left of the form, there is a blue button labeled '+ Add Argument'.

1. Click on the **Functions** link.
2. Click the add function button (+ **Add Function**).
3. Replace the random function name with a function name that makes sense for the function you're defining such as **used-percentage**.
4. In the **Path to function** field, enter the name of the python file that contains the functions. These files must be stored in the `../healthbot/input/` directory. The pull-down list is populated with all of the python (.py) files in that directory.
5. In the **Method name** field, enter the name of the method as defined in the python file, for example `getmpllabel`.

6. (Optional) Enter a description for the function
7. (Optional) For each argument that the python function can take, click the add argument button (**+ Add Argument**).

Each time you click the add argument button, you'll need to enter the name of the argument and set the toggle switch as to whether the argument is mandatory or not. The default is that none of the arguments are mandatory.

Triggers

A required element of rule definition that you'll need to set is the trigger element. [Figure 46 on page 133](#) shows the **Triggers** block for the `system.memory/check-system-memory` rule. The field details are discussed below.

Figure 46: The Triggers Block

The screenshot displays the 'Triggers' configuration panel. At the top, there are navigation tabs: Sensors, Fields, Vectors, Variables, Functions, **Triggers**, and Rule Properties. A '+ Add Trigger' button is on the left. The main area shows a trigger named 're-memory-buffer-utilization' with a 'Delete re-memory-buffer-utilization' button. Below the name is a 'Frequency' field with the placeholder 'Enter trigger frequency'. A dropdown menu shows the selected term 'is-re-memory-buffer-utilization'. The 'WHEN' section has four columns: 'Left operand' ('\$re-memory-buffer'), 'Operator' ('>='), 'Right operand' ('\$re-memory-buffer-high-threshold'), and 'Time range' ('5m'). An '+ Add Condition' button is below. The 'THEN' section has a 'Color' dropdown (red) and a 'Message' text area containing 'Routing-engine memory buffer utilization(\$re-memory-buffer) exceed high threshold(\$re-memory-buffer-high-threshold)'. An 'Evaluate next term' toggle is below the message. An information box at the bottom states: 'Functions can be used as Trigger actions too, define them using the 'Functions' menu at the top.'

Setting up triggers involves creating terms that are used to set policy. If the terms within a trigger are matched, then some action is taken. Terms can evaluate the fields, functions, variables, etc that are defined within the rule against each other or in search of specific values. Terms are evaluated in order from the top of the term list to the bottom. If no match is found, then the next term (if any) is evaluated until a match is found or until the bottom of the term stack is reached.

1. Click on the **Triggers** link.
2. Click on the add trigger button (+ **Add Trigger**).
3. Replace the random trigger name with one that makes sense for the trigger you are defining, such as **foo-link-operation-state**. We recommend using a name that is very unique to the rule and trigger to avoid having the same trigger name across two or more rules.

4. (Optional) Enter a value in the **Frequency** field. This value tells HealthBot how often the field data and triggers should be queried and evaluated. If no entry is made here, then the sensor frequency is applied for this value. The frequency entered here can be entered as a multiple or, an offset, of the sensor frequency such as 2o. For example, if the sensor frequency is 10s and the trigger frequency is 2o, then the trigger frequency would be 20s (2*10s).

5. Click the add term button (+ **Add Term**).

The Term area will expand and show an add condition button, (+ **Add Condition**) in the **When** section and **Color** and **Message** fields in the **Then** section.

6. To define a condition that the term will evaluate, click the + **Add Condition** button.

The **When** section expands to show **Left operand**, **Operator**, and **Time range** fields.

NOTE: Setting a condition is not required. If you want to guarantee that a **Term** takes a specific action, don't set a condition. This could be useful, for example, at the bottom of a term stack if you want some indication that none of the terms in your trigger matched.

7. Select values from the pull-down menus for each of these fields.

Depending on which **Operator** is chosen, a new field, **Right operand** may appear in between the **Operator** and **Time range** fields.

The left and right operand pull-down menus are populated with the fields and variables defined in the rule. The operator field determines what kind of comparison is done. The time range field allows the trigger to evaluate things such as if there were any dropped packets in the last minute.

8. (Optional) Set values for the **Color** and **Message** fields of the **Then** section.

These fields are the action fields. If a match is made in the condition set within the same term, then whatever action you define here is taken. A color value of green, yellow, or red can be set. A message can also be set and is not dependent on whether any color is set.

If color or message are set, a toggle button labelled **Evaluate next term** appears at the bottom of the **Then** section. The default value for this button is off (not active).

NOTE: If no match is made in the **When** section of a term, the **Then** section is ignored. If this happens, the next term down, if any, is evaluated.

If a match is made in the **When** section, then the actions in the **Then** section, if any, are taken and processing of terms stops unless the **Evaluate next term** button is set to on (active).

Setting the **Evaluate next term** button allows you to have HealthBot make more complex evaluations like 'if one condition and another condition are both true, then take this action'.

Rule Properties

(Optional) Specify metadata for your HealthBot rule in the **Rule Properties** block. Available options include:

Attributes	Description
Version	Enter the version of the HealthBot rule.
Contributor	Choose an option from the drop-down list.
Author	Specify a valid e-mail address.
Date	Choose a date from the pop-up calendar.
Supported HealthBot Version	Specify the earliest HealthBot release for which the rule is valid.
Supported Device > Juniper Devices	Choose either Junos or Junos Evolved. Device metadata includes Product Name, Release Name, Release Support (drop-down list), and platform. You can add metadata for multiple devices, multiple products per device, and multiple releases per product.
Supported Device > Other Vendor Devices	Specify information about non-Juniper devices. You can add metadata for multiple devices.
Helper Files	Specify files that are required by the HealthBot rule.

Edit a Rule

To edit a rule:

1. Click the **Rules** icon in the left-nav bar.
2. Click the name of the rule listed along the left side of the Rules page.
3. Modify the necessary fields.
4. Select one of the following options:

Save	Save your edits 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.
Save & Deploy	Immediately deploy the configuration and save the rule.

5. (Optional) Delete a rule by clicking the **Delete** button located to the right of the rule name.

Add a Pre-Defined Playbook

Juniper curates a set of pre-defined playbooks designed to address common use cases. You can add these playbooks to your HealthBot installation at any time. The default pre-defined playbooks cannot be changed or removed.

To add a pre-defined playbook to HealthBot:

1. Using a browser, go to <https://github.com/Juniper/healthbot-rules> and download the pre-defined playbook file to your computer.
2. In the HealthBot GUI, click the **Configuration > Playbooks** icon in the left-nav bar.
3. Click the upload playbook button (↑ **Upload Playbook**).
4. Click the **Choose Files** button.
5. Navigate to the playbook file and click **Open**.
6. Select one of the following options:

Upload	Upload the file and add the playbook 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.
Upload & Deploy	Upload the file, add the playbook, and immediately deploy the configuration.

Create a New Playbook Using the HealthBot GUI

HealthBot operates on playbooks, which are a collection of rules for solving a specific customer use case. For example, the system-kpi-playbook monitors the health of system parameters such as system-cpu-load-average, storage, system-memory, process-memory, etc. and notifies the operator or takes corrective action in case any of the KPIs cross pre-set thresholds. Any single rule can be a part of 0, 1, or more playbooks. The playbook is the rule element that gets deployed on devices. Rules that are not included in any playbook will not be deployed to any device.

NOTE: Click the **Name**, **Running**, **Paused**, or **Synopsis** column headers in the Playbooks table to organize the data in ascending or descending order.

To create a new playbook using the HealthBot GUI:

1. Click the **Configuration > Playbooks** icon in the left-nav bar.
2. Click the create playbook button (+ **Create Playbook**).

A new window appears with 4 fields: Name, Synopsis, Description, and Rules. We describe the use of each field below.

3. Enter a name for the playbook in the **Name** field.
4. Enter a short description for the playbook in the **Synopsis** field.
This text appears in the Synopsis column of the table on the **Playbooks** page.
5. (Optional) Enter a description of each of the rules that make up this playbook in the **Description** field.
This text can only be seen if you click on the playbook name on the **Playbooks** page.
6. From the rules drop-down list, select the rules that make up this playbook.
7. Select one of the following options:

Save	Save and add the playbook 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.
Save & Deploy	Immediately deploy the configuration and save and add the playbook.

Edit a Playbook

To edit a playbook:

1. Click the **Configuration > Playbooks** icon in the left-nav bar.
2. Click the name of the playbook.
3. Modify the necessary text boxes.
4. Select one of the following options:

Save	Save your edits to the playbook 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.
Save & Deploy	Immediately deploy the configuration and save your edits to the playbook.

5. (Optional) Delete a playbook by clicking the trash can icon in the **Delete** column.

NOTE: You cannot edit or delete a system defined (Juniper provided) playbook.

Manage Playbook Instances

IN THIS SECTION

- [View Information About Playbook Instances | 141](#)
- [Create a Playbook Instance | 143](#)
- [Manually Pause or Play a Playbook Instance | 145](#)
- [Create a Schedule to Automatically Play/Pause a Playbook Instance | 146](#)

The term *playbook instance* refers to a specific snapshot of a playbook that is applied to a specific device group or network group. You can manually play and pause playbook instances. Alternatively, you can apply a customized schedule to a playbook instance that will automatically perform play and pause actions.

The following sections describe tasks that you can perform to manage playbook instances:

View Information About Playbook Instances

To view information about playbook instances:

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

The saved playbooks are listed in the table on the main Playbooks page.

Playbook	Instances		Action			Synopsis
	Name	Running	Paused	Apply	Live	
bgp-route-hijack-detection	0	0	↗	●	🗑️	Playbook detects route hijack
bgp-session-stats-playbook	0	0	↗	●	🗑️	BGP neighbor sessions key performance indicators
chassis-kpis-playbook	0	0	↗	●	🗑️	Chassis key performance indicators
chip-agnostic-kpis	0	0	↗	●	🗑️	Chip agnostic kpis
DHCP-server-statistics	0	0	↗	●	🗑️	DHCP Local Server and Relay Statistics KPIs
dot1x-user-authentication-kpis	0	0	↗	●	🗑️	dot1x authentication KPI
evpn-irb-icmp-probe	0	0	↗	●	🗑️	EVPN-VXLAN rvi key performance indicators
evpn-vxlan-kpis	0	0	↗	●	🗑️	EVPN-VXLAN key performance indicators
forwarding-table-summary	0	0	↗	●	🗑️	Forwarding table and protocol routes key performance indicators
get-vpn-stats <small>Requires advanced licensing</small>	0	0	↗	●	🗑️	Interface and routing instance collector
icmp-outlier	0	0	↗	●	🗑️	ICMP outlier detector
icmp-probe	0	0	↗	●	🗑️	ICMP RTT response checker
▶ interface-kpis-playbook	1	0	↗	●	🗑️	Interface key performance indicators
interface-optical-kpis	0	0	↗	●	🗑️	Optical interface key performance indicators
isis-stats-playbook	0	0	↗	●	🗑️	ISIS adjacency key performance indicators

- Playbooks that have been applied to a device group or network group are identified in the table by a right caret next to the playbook name.
- The Instances column in the table shows the number of playbook instances running and paused.
- Starting with HealthBot 3.1.0, some playbooks require the purchase and installation of advanced or premium licenses. These playbooks are identified by the green circle with a white star in it. As shown above, it tells you which license is required when you hover your mouse over the icon.
- The Live column (in the Action section of the table) shows a colored circle indicator that represents the overall status of the playbook instances for each playbook. The following table provides the color definitions:

Table 5: Color Definitions for the Live Column

Color	Definition
Green	All instances associated with this playbook are currently running.
Yellow	One or more instances associated with this playbook are paused.
Gray/Black	There are no instances associated with this playbook, or an instance is associated with this playbook but the configuration has not been deployed yet.

- Click on the caret next to the playbook name to expand or collapse the playbook instance details. If no caret is present, then the playbook has not been applied to any device groups or network groups.

The following playbook instance details are displayed:

Column Name or Widget	Description
Instance Name	User-defined instance name.
Schedule	Name of the schedule profile applied to the playbook instance. For information on how to configure a schedule profile, see “Create a Schedule to Automatically Play/Pause a Playbook Instance” on page 146 Click on the name to display the schedule details.
Device/Network Group	Device group or network group to which the schedule is applied.
No. of devices	Number of devices on which this playbook instance is deployed. This is applicable for device group instances only, not for network group instances.
Status	Current status of the playbook instance. Status can be either Running or Paused. The Status column also indicates whether the action was performed automatically or manually. Note: If the status of a playbook instance is Running (automatic) , you can manually pause the schedule for this instance using the Pause Schedule button. In this case, the status will change to Paused (manual) . To resume running the schedule for this instance, you must manually run the instance using the Play Schedule button. In this case, the status will change back to Running (automatic) , and the play and pause actions associated with the schedule will resume.
Started/Paused at	Date and time when the playbook instance was last started or paused. The date reflects local browser time zone.

Column Name or Widget	Description
Next Action	This column applies only to playbook instances associated with a schedule. It indicates whether the playbook instance is scheduled to automatically pause or play in the future. This column is blank if no schedule is associated with the playbook instance or if the status of the instance is Paused (manual) .
Play/Pause button	Pauses or plays a playbook instance or the schedule for a playbook instance. The Play/Pause button toggles between the two states. For more information, see “Manually Pause or Play a Playbook Instance” on page 145 .
Trash can icon	Deletes the playbook instance.

Create a Playbook Instance

To create a playbook instance for a device group:

1. Click the **Configuration > Playbooks** option in the left-nav bar.
2. Click the **Apply** icon (in the Action section of the table) for the desired playbook.
A pane titled *Run Playbook: <playbook-name>* appears.
3. In the **Name of Playbook Instance** field, fill in an appropriate name for this instance of the playbook. This is a required field.
4. (Optional) In the **Run on schedule** field, choose the name of the schedule that you want to apply to this playbook instance. You can apply only one schedule per playbook instance. If you want a specific playbook instance to run on multiple schedules, you must create multiple versions of the instance, each with its own unique name and schedule.

For information on how to configure a schedule, see [“Create a Schedule to Automatically Play/Pause a Playbook Instance” on page 146](#).

To view information about an existing schedule:

- a. Click the **Settings** option in the left-nav bar.
 - b. In the **Scheduler Settings** section, a summary of the properties for each saved schedule is shown in the table. Click on a specific schedule name to view additional details.
5. In the **Device Group** section under **Rules**, apply this playbook instance to the appropriate device group using the drop-down list.

The list of devices in the **Devices** section changes based on the device group selected.

NOTE: If your playbook contains network rules, the **Device Group** section does not appear. Instead, it is replaced with a **Network Groups** section (not shown).

- Click one of the devices listed in the **Devices** section.

Here is where you can customize the variable values that will be set for this device when the playbook is run.

- In the section titled **Variable values for Device <Device Name>**, the variables for each rule in the playbook can be seen by clicking on the rule name. The default values for each variable are displayed as grey text in each field. You can leave these values as-is or override them by entering a new value.

Repeat steps 6 and 7 for each device in the device group, as needed.

- When you are satisfied that all of the variable values are appropriate for all the devices in the device group, select one of the following options.

Save Instance	Save your edits but do not deploy the updated configuration and do not run the instance. You can use this option when, for example, you are making several changes and want to deploy all your updates at the same time.
Run Instance	Deploy the configuration, and, if no schedule profile was applied, immediately run the instance. If a schedule profile was applied, the instance will run according to the configuration of the profile.

Manually Pause or Play a Playbook Instance

When an instance is paused, HealthBot does not collect, analyze or raise an alarm on the device or network group data associated with the playbook rules. Data collected prior to pausing the instance is retained in the system, but no new data is collected or analyzed until the instance is played again.

The following table describes the state of the playbook instance when a particular button is displayed in the Play/Pause column:

If the displayed Play/Pause button is...	Then the state of the playbook instance is...
Pause Instance	<ul style="list-style-type: none"> • The instance is running. • The instance is not associated with a schedule. • Data is being collected.
Pause Schedule	<ul style="list-style-type: none"> • The instance is associated with a schedule. • The schedule is running. • The schedule determines when the instance is running.
Play Instance	<ul style="list-style-type: none"> • The instance is not running. • The instance is not associated with a schedule. • No data is being collected.
Play Schedule	<ul style="list-style-type: none"> • The instance is associated with a schedule. • The schedule and the instance is not running. • No data is being collected.

To manually pause a playbook instance or schedule:

1. Click the **Configuration > Playbooks** option in the left-nav bar.
The list of existing playbooks is displayed.
2. Click the caret next to the name of the playbook that you want to pause.
3. Choose one of the following options:
 - Click the **Pause Instance** button to pause a playbook instance (not associated with a schedule).
 - Click the **Pause Schedule** button to pause the schedule that's associated with a playbook instance.
4. The Play/Pause Playbook Instance dialog box appears. Select one of the following options:

Pause	Flags this playbook instance to be paused the next time you deploy the configuration. Use this option if you are making several changes and want to deploy all your edits at the same time.
--------------	---

Pause & Deploy

Immediately pause the playbook instance and deploy the configuration. It will take a few seconds for the playbook table to update to show the instance is paused.

There is a slight delay in updating the table because the play and pause actions are asynchronous and run in the background, allowing you to perform other actions. The status of this asynchronous activity can be tracked through the deploy icon located in the upper right corner of the window (as indicated in the success message of deploy action). Once this action is complete, the status is reflected in the playbook table as well.

Once the playbook table is refreshed, the playbook name shows a yellow icon in the Live column as a visual indicator that an instance is paused.

5. To resume a paused playbook instance, follow the same steps as above except choose one of the following options for Step 3:
 - Click the **Play Instance** button to resume running a playbook instance (not associated with a schedule).
 - Click the **Play Schedule** button to resume running the schedule associated with a playbook instance. The schedule determines when the instance resumes playing.

Create a Schedule to Automatically Play/Pause a Playbook Instance

To automatically play/pause a playbook instance, you must first create a schedule and then apply the schedule to the playbook instance. You can apply only one schedule per playbook instance. If you want a specific playbook instance to run on multiple schedules, you must create multiple versions of the instance, each with its own unique name and schedule.

To create a schedule for a playbook instance:

1. Click the **Settings > System** option in the left-nav bar.
2. Click the **Scheduler** tab.
3. In **Scheduler Settings**, click the add scheduler button (+ **Scheduler**).
4. Enter the necessary values in the text boxes and select the appropriate options for the playbook instance schedule.

The following table describes the attributes in the **Add a scheduler** and **Edit a scheduler** panes:

Attributes	Description
Name	Enter the name of the playbook instance schedule.

Attributes	Description
Scheduler Type	<p>Choose discrete.</p> <p>You can configure a discrete length of time to play the playbook instance using the Run for field. Once the run time has ended, HealthBot will automatically pause the instance. You can also configure HealthBot to automatically resume playing the instance using the Repeat field.</p>
Start On	<p>Use the pop-up calendar to select the date and time to play the playbook instance for the first time.</p>
Run for	<p>Configure a discrete length of time to play the playbook instance. First enter an integer value and then choose the unit of measure (minute, hour, or day) from the drop-down list.</p> <p>Once the run time has ended, HealthBot will automatically pause the instance. You can also configure HealthBot to automatically resume playing the instance using the Repeat field.</p>
End On	<p>(Optional) Use the pop-up calendar to select the date and time to pause the playbook instance indefinitely. Leave blank if you want the playbook instance to play indefinitely.</p> <p>Note: If a playbook instance is associated with a schedule and is running when the End On time is reached, then the instance will continue to run until the configured Run for length of time is reached. At this time, the instance will pause indefinitely.</p>
Repeat	<p>Configure the Run for field before configuring the Repeat field. The Repeat interval must be larger than the configured Run for length of time.</p> <p>In the drop-down list, choose one of the following:</p> <ul style="list-style-type: none"> ● The frequency (every day, week, month, or year) at which you want the playbook instance to play. ● The Never option if you want the playbook to play only once. ● The Custom option to specify a custom frequency at which you want the playbook instance to play. Use the Repeat Every field to configure the custom frequency.
Repeat Every	<p>(Optional) If you chose the Custom option for the Repeat field, enter the custom frequency at which you want the playbook instance to play. First enter an integer value and then choose the unit of measure (minute, hour, or day) from the drop-down list.</p>

5. Click **Save** to save the configuration or click **Save and Deploy** to save and deploy the configuration.
6. Now you're ready to apply the schedule to a playbook instance. For more information, see ["Create a Playbook Instance"](#) on page 143.

Release History Table

Release	Description
3.1.0	Starting in HealthBot Release 3.1.0, you can add fields and keys to rules based on whether the incoming data meets user-defined conditions defined in tagging profiles.
3.1.0	Starting with HealthBot 3.1.0, some playbooks require the purchase and installation of advanced or premium licenses.
3.0.0	Starting in HealthBot Release 3.0.0, you can filter the Topics and Rules displayed on the left side of the Rules page.

RELATED DOCUMENTATION

[HealthBot Concepts | 20](#)

[Manage Devices, Device Groups, and Network Groups | 104](#)

Monitor Device and Network Health

IN THIS SECTION

- [Dashboard | 149](#)
- [Health | 152](#)
- [Network Health | 165](#)
- [Graph Page | 165](#)

HealthBot offers several ways to detect and troubleshoot device-level and network-level health problems. Use the information provided by the following HealthBot GUI pages to investigate and discover the root cause of issues detected by HealthBot:

Dashboard

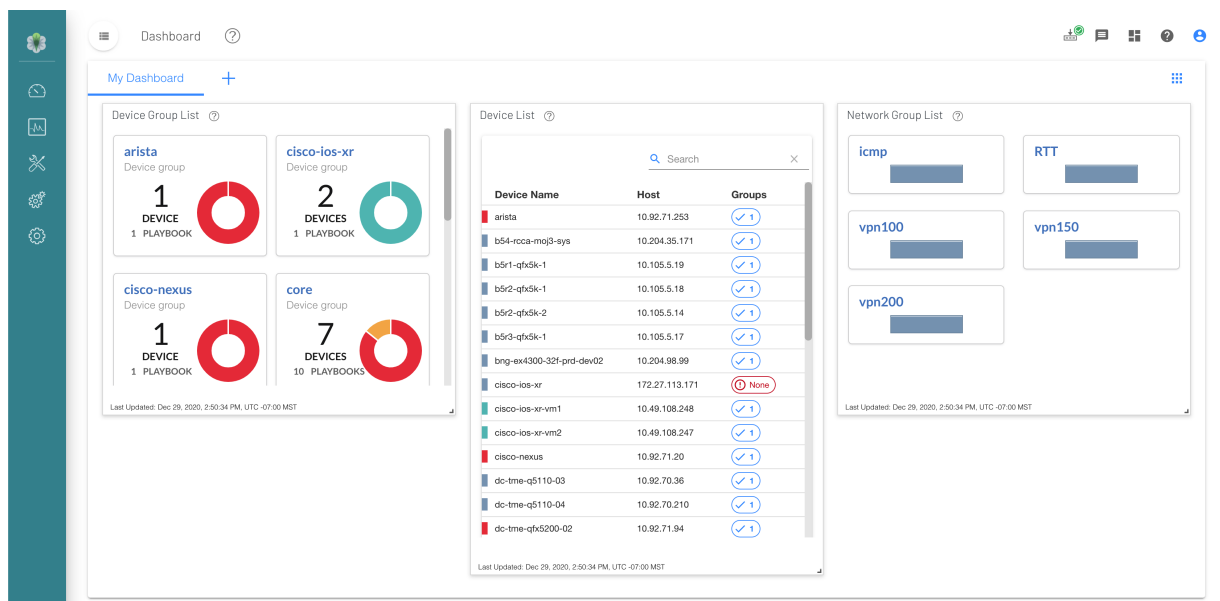
Use the **Dashboard** to create a custom view of what you're most interested in. HealthBot pre-populates the dashboard with the **Device List**, **Device Group List**, and **Network Group List** dashlets and calls this view **My Dashboard**. You can create your own dashboard view by clicking the + to the right of **My Dashboard**. Custom views can be added, renamed, and deleted as you see fit.

The **Dashboard** also has a graphical list of pre-defined dashlets across the top that is initially hidden from view. Click the cluster of 9 blue dots on the upper right part of the page to display or hide the available dashlets. Each dashlet provides graphical information from a specific point of view. Many of the dashlets can be clicked on to drill down deeper into the information presented.

- **Device Group List**—Consists of device group dashlets (see “[Device Group List Dashlet](#)” on page 151).
- **Devices**—Consists of a table that lists all of the HealthBot devices (see “[Device List Dashlet](#)” on page 150).
- **Network Groups**—Consists of network group dashlets (see “[Network Group List Dashlet](#)” on page 152).

A customized Dashboard view is shown in [Figure 47 on page 149](#).

Figure 47: HealthBot Dashboard



The Dashboard uses two types of colored objects to provide health status: halos and bars. The following table describes the meaning of the severity level colors displayed by the status halos and bars on the Dashboard:

Color	Definition
Green	The overall health of the device, device group, or network group is normal. No problems have been detected.
Yellow	There might be a problem with the health of the device, device group, or network group. A minor problem has been detected. Further investigation is required.
Red	The health of the device, device group, or network group is severe. A major problem has been detected.
Gray	No data is available.

Device List Dashlet

The following table describes the main features of the Devices List dashlet on the Dashboard. For information on how to add a new device, see [“Manage Devices, Device Groups, and Network Groups” on page 104](#).

Feature	Description
Edit the device properties.	Click the device name. For information on device properties, see “Manage Devices, Device Groups, and Network Groups” on page 104 .
Filter the devices based on a device name or hostname/IP address.	At the top of the table, enter the device name, hostname, or IP address in the search field. The list updates as you type.
Refresh the data in the dashlet	Click the circular arrow at the top of the dashlet.
Delete the dashlet	Click the X at the top of the dashlet

Device Group List Dashlet

The following table describes the main features of the device group list dashlet on the Dashboard. For information on how to add a new device group, see [“Manage Devices, Device Groups, and Network Groups” on page 104](#).

Feature	Description
Edit the device group properties.	Click the device group name. For information on device group properties, see “Manage Devices, Device Groups, and Network Groups” on page 104 .
Display the list of devices that belong to a device group, as well as a health status bar for each device	Click the integer number on the Device Group dashlet that represents the number of devices included in the device group.
Display the list of playbooks to which the device group is applied.	Click the line that shows playbook count on the dashlet.
Delete the device group dashlet.	Click the X button on the device group dashlet.
Status halo	The coloring of the status halo represents the percentage of devices in the device group that have the health status defined by the color. For example, if the circle is all green, then the health of 100% of the devices in the device group is normal.
Status bar	The color of the status bar represents the overall health status of the device in the device group. Clicking the bar takes you to the Monitor > Device Group Health page, filtered for the specific device you clicked.

Network Group List Dashlet

The following table describes the main features of the network group dashlet on the Dashboard. For information on how to add a new network group, see [“Manage Devices, Device Groups, and Network Groups” on page 104](#).

Feature	Description
Edit the network group properties.	Click the network group name. For information on network group properties, see “Manage Devices, Device Groups, and Network Groups” on page 104 .
Delete the network group dashlet.	Click the X at the top of the dashlet.
Open the Network Health page for a specific network group.	Click the status bar on a network group in the dashlet.
Status bar	The color of the status bar represents the overall health status of the network group.

Health

Use the Health page (**Monitor > Health**) to monitor and track the health of a single device, a device group, or a network. You can also troubleshoot problems. Select a device group using the entity type selectors (DEVICE, DEVICE GROUP, or NETWORK) located in the top left corner of the page. Once selected, you can then select individual devices or all of the devices from the group by clicking the **Select devices** pull-down menu. The page is divided into the following three main views that, when used together, can help you investigate the root cause of problems detected on your devices:

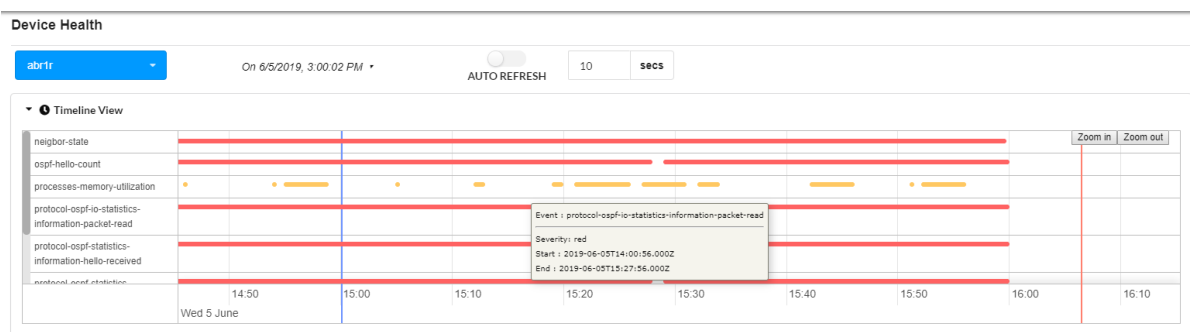
- [Timeline View on page 153](#)
- [Tile View on page 156](#)
- [Table View on page 159](#)
- [Time Inspector View on page 162](#)

Timeline View

In timeline view, you can monitor real-time and past occurrences of KPI events flagged with a minor or major severity level health status. The general characteristics and behaviors of the timeline include (see [Figure 48 on page 154](#)):

- Clicking on the right caret next to the Timeline View heading expands or collapses the timeline.
- Each dot or line in the timeline represents the health status of a unique KPI event (also known as a HealthBot rule trigger) for a pre-defined KPI key with which HealthBot has detected a minor or major severity level issue. The name of each event is displayed (per device) directly to the left of its associated health status dot or line.
- The health status dot or line for each unique KPI event in the timeline can consist of several different KPI keys. Use tile view and table view to see the health status information for the individual KPI keys.
- Only minor or major severity level KPI events are displayed in the timeline. Yellow represents a minor event, and red represents a major one.
- A KPI event that occurs once (at only one point in time) and does not recur continuously over time is represented as a dot.
- A KPI event that occurs continuously over time is represented as a horizontal line.
- Timeline data is displayed for a 2-hour customizable time range.
- The red vertical line on the timeline represents the current time.
- The blue vertical line on the timeline represents the user-defined point of time for which to display data.

Figure 48: Timeline view



The following table describes the main features of the timeline:

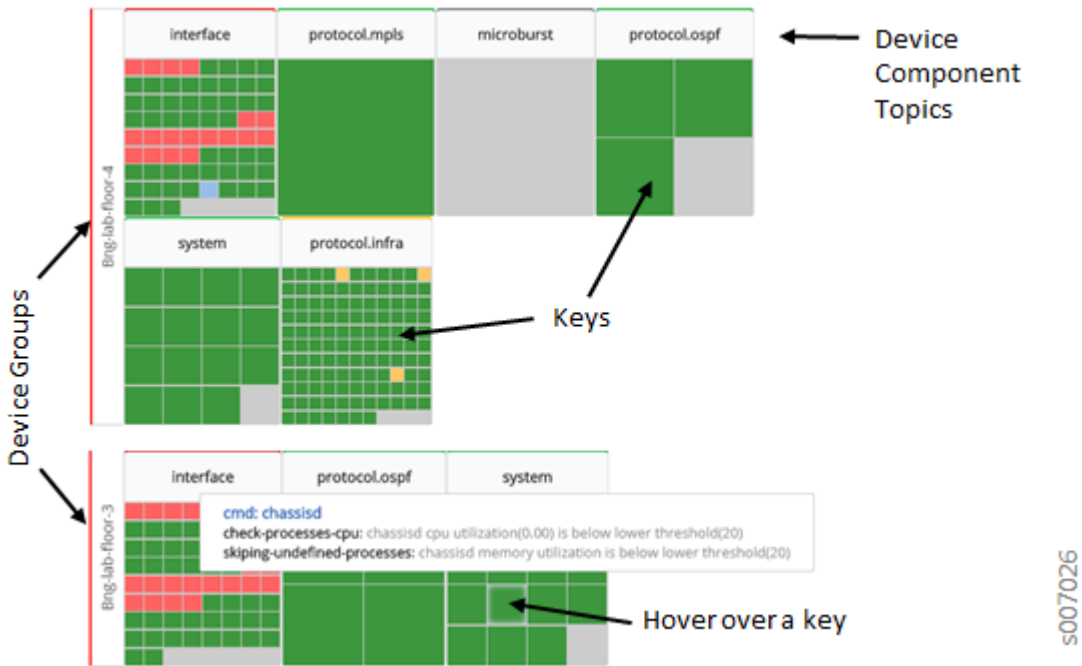
Feature	Description
Display information about a dot or horizontal line in the timeline.	<p>Hover over the dot or horizontal line to display the associated KPI event name, device name, health status severity level, and event start and end times.</p> <p>Additional health status information about the KPI event can be found in tile view. For information about tile view, see the “Tile View” on page 156 section.</p>

Feature	Description
For the displayed data, change the range of time (x-axis) that is visible on the page.	<p>Options:</p> <ul style="list-style-type: none"> • Click and drag the x-axis of the timeline to the left or to the right. • Click the Zoom In or Zoom Out buttons in the top right corner of the timeline.
Choose a different 2-hour time range of data to display.	<p>Use the blue vertical line to customize the time range of data to display. Options for enabling the blue vertical line:</p> <ul style="list-style-type: none"> • Click inside the timeline grid at the particular point in time you want to display data. • In the date/time drop-down menu (located above the timeline), select the particular point in time you want to display data .. <p>Data is generally displayed for 1 hour before and 1 hour after the blue line. Hover over the blue line to display the exact point in time that it represents. Drag the blue line left or right to adjust the time.</p> <p>NOTE: Auto-refresh is disabled whenever you enable the blue line. Re-enabling auto-refresh disables the blue line and resets the timeline to display the most recent 2-hour time range of data.</p>
Freeze the timeline (disable auto-refresh).	Toggle the auto-refresh switch to the left.
Unfreeze the timeline (enable auto-refresh).	Toggle the auto-refresh switch to the right.

Tile View

The tile view uses colored tiles to allow you to monitor and troubleshoot the health of a device. The tiles are organized first by device group, then by device component topic, and lastly by unique KPI key (see [Figure 49 on page 157](#)). By default, the tile view data corresponds to the most recent data collected. To customize the point in time for which data is displayed in tile view, select a particular point in time from the date/time drop-down menu (located above the timeline) or enable the blue vertical line in timeline view. For information about how to enable the blue vertical line, see the [“Timeline View” on page 153](#) section. The **Composite** toggle switch (not shown) at the upper right of the **TILE VIEW**, allows you to select data from more than one device component topic to be shown in the **Table View** and, thus, the Time Inspector View. This can be useful when topics must be combined to find root cause for an issue. For example, system memory usage could combine with output queue usage to create a performance issue in an overloaded system.

Figure 49: Tile View



The following table describes the meaning of the severity level colors displayed by the status tiles:

Color	Definition
Green	The overall health of the KPI key is normal. No problems have been detected.
Yellow	There might be a problem with the health of a KPI key. A minor problem has been detected. Further investigation is required.
Red	The health of a KPI key is severe. A major problem has been detected.

Color	Definition
Gray	No data is available.

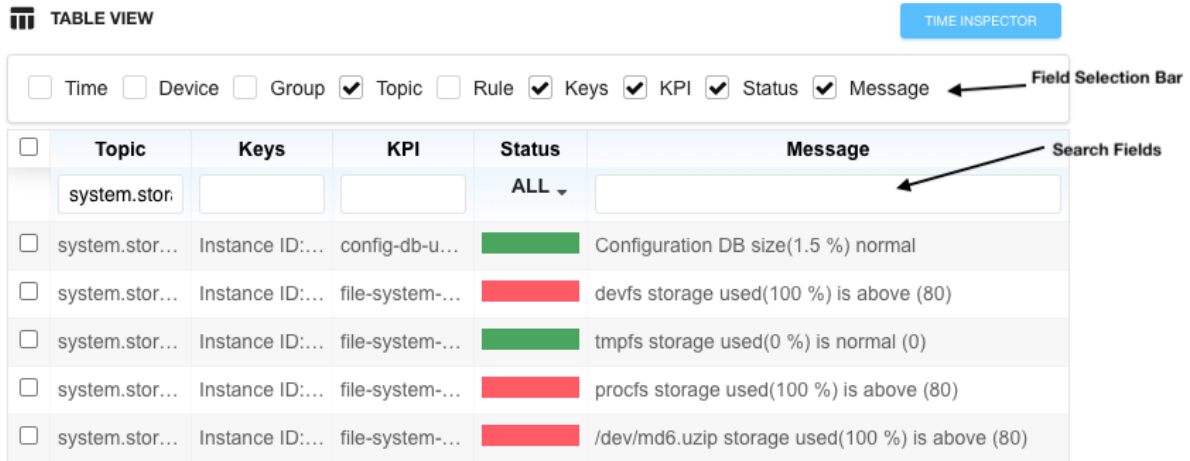
The following table describes the main features of the tile view:

Feature	Description
Display information about a status tile.	<p>Options:</p> <ul style="list-style-type: none"> • Hover over a status tile to display the name of the key, KPIs associated with the key, and the status messages associated with the KPIs. • Click on a status tile. Information about the status tile is displayed in table view. For information about table view, see the “Table View” on page 159 section. <p>Note: If the number of KPI keys exceeds 220, the keys are automatically aggregated and grouped.</p>
Display information in table view about the status tiles associated with a single device component topic.	Click on a device component topic name in tile view. For information about table view, see the “Table View” on page 159 section.
Composite Toggle	When active, users can click on specific keys within the tile groups. This allows you to pass multiple KPIs to the Time Inspector View.

Table View

The table view allows you to monitor and troubleshoot the health of a single device based on HealthBot data provided in a customizable table. You can search, sort, and filter the table data to find specific KPI information, which can be especially useful for large network deployments. To select which attributes are displayed in the table, check the appropriate check box in the field selection bar above the table (see [Figure 50 on page 160](#)). The checkbox on the left side of each row is used to help activate the Time Inspector view. Multiple rows can be selected at one time.

Figure 50: Table View



The following table describes the HealthBot attributes supported in table view:

Attributes	Description
Time	Time and date the event occurred.
Device	Device name.
Group	Device group name.
Topic	Rule topic name.
Keys	Unique KPI key name.
KPI	Key Performance Indicator (KPI) name associated with an event.
Status	Health status color. Each color represents a different severity level.
Message	Health status message.

The following table describes the meaning of the severity level colors displayed by the **Status** column:

Color	Definition
Green	The overall health of the KPI key is normal. No problems have been detected.
Yellow	There might be a problem with the health of a KPI key. A minor problem has been detected. Further investigation is required.
Red	The health of a KPI key is severe. A major problem has been detected.
Gray	No data is available.

The following table describes the main features of the table view:

Feature	Description
Sort the data by ascending or descending order based on a specific data type.	Click on the name of the data type at the top of the column by which you want to sort.
Filter the data in the table based on a keyword.	Enter the keyword in the text box under the name of a data type at the top of the table (see Figure 50 on page 160).
Navigate to a different page of the table.	Options: <ul style="list-style-type: none"> • At the bottom of the table, click the Previous or Next buttons. • At the bottom of the table, select the page number using the up/down arrows (or by manually entering the number) and then press Enter.
If the data in a cell is truncated, view all of the data in a cell.	Options: <ul style="list-style-type: none"> • Hover over the cell. • Resize the column width of the cell by dragging the right side of the title cell of the column to the right.
Row selection checkbox	Make this row's data available for Time Inspector view.

Time Inspector View

Starting with HealthBot Release 3.2.0, a new view called Time Inspector is available. Time Inspector is a composite view that is available only when the entity type **DEVICE GROUP** is selected. It can be accessed by clicking the **TIME INSPECTOR** button located below the **Timeline View** pull-down menu.

This view allows you to drill down into which specific triggers within a given rule caused a sensor to display green, yellow, or red status.

When the **Health** page is first accessed, the **TIME INSPECTOR** button is disabled. To activate the button and make the view available, you must:

- Select the **Entity Type DEVICE GROUP** in the top section of the Health page.
- Select at least one device from the **Devices** pull-down menu.
- Have valid data in at least one device component topic in **TILE VIEW**.

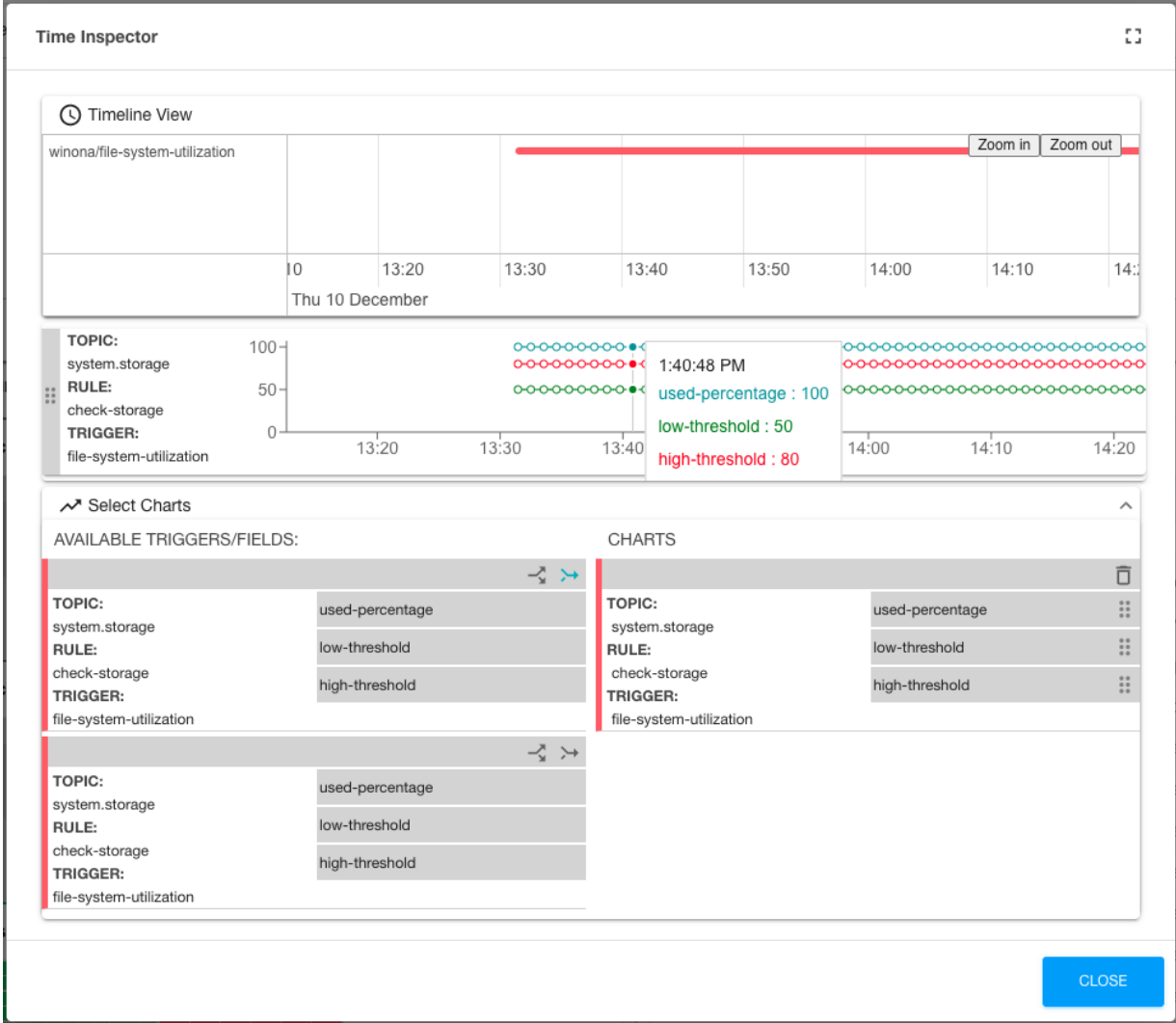
NOTE: Topics showing “no data” will not work for enabling the Time Inspector view.

- Have data appearing in the **TABLE VIEW** section. You can achieve this by clicking the device component topic header in **TILE VIEW**.
- Select the checkbox to the left of at least one of the rows in **TABLE VIEW**.

When clicked, the **TIME INSPECTOR** button opens a pop-up window above the Health page.

[Figure 51 on page 163](#) below shows a time inspector window created from the system.storage usage topic for a specific device.

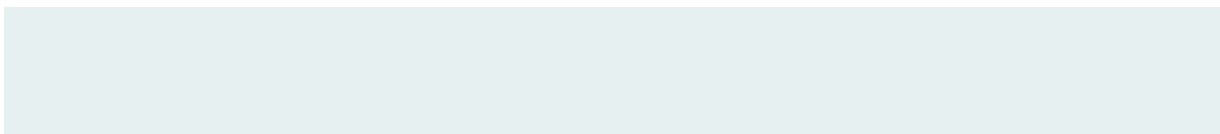
Figure 51: Time Inspector Window



As you can see, the Time Inspector window has a mini timeline at the top, an incremented line chart below, and a chart selector section at the bottom. This particular chart was created as a composite (indicated by the merging blue arrow) of a file-system-utilization in the check-storage rule of the system.storage topic.

Note that there are 3 fields in the check-storage rule: used-percentage, low-threshold, and high-threshold. Since the chart was created as a composite (fields charted together) there are three lines on the displayed chart. If the “chart fields separately” button (diverging arrows) were clicked instead, you would see 3 single-line charts showing the same data.

The more rules you select with the **TABLE VIEW** checkboxes, the more charts you can create in the **Time Inspector** view.



Network Health

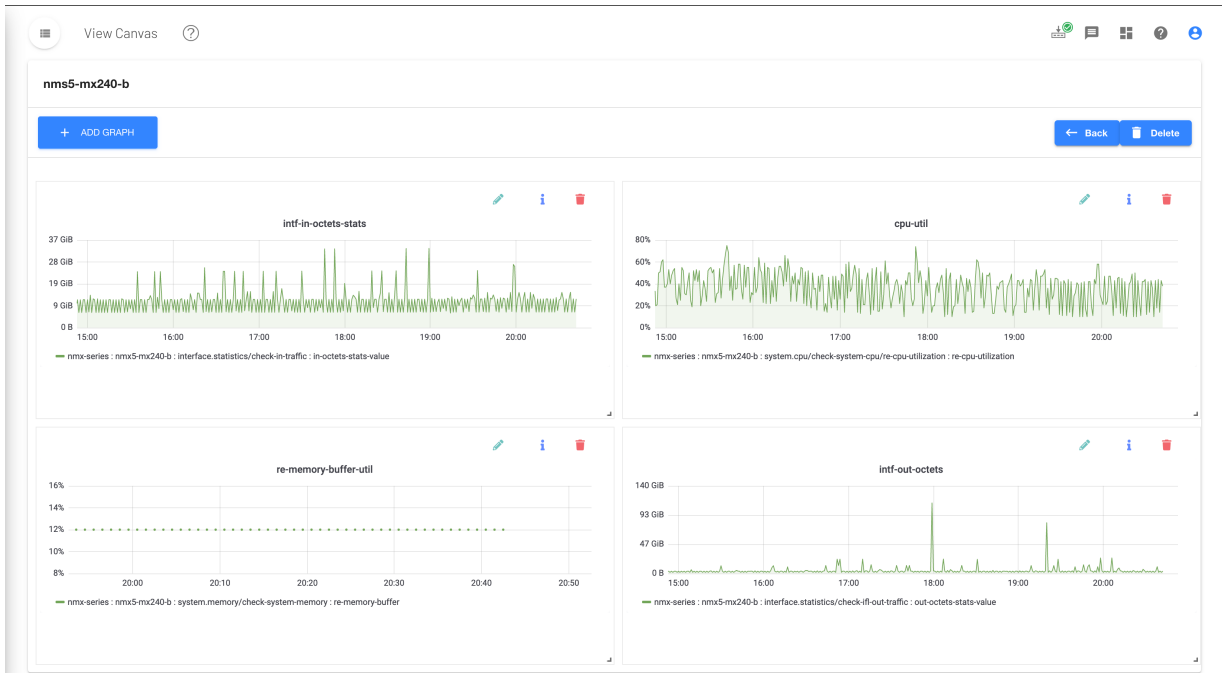
Use the Network Health page (**Monitor > Network Health**) to monitor and track the health of a Network Group and troubleshoot problems. Select a Network Group using the drop-down list located in the top left corner of the page. Comparable to the Device Group Health page (see the [“Health” on page 152](#) section), the Network Health page is divided into three main views: timeline, tile, and table. The Network Health page provides similar features and functionality for a network group as the Device Group Health page provides for a single device.

Graph Page

You can use graphs to monitor the status and health of your network devices. Graphs allow you to visualize data collected by HealthBot from a device, showing the results of rule processing. Access the page from the left-nav panel **Monitor > Graph**

NOTE: Graphs are refreshed every 60 seconds.

Figure 52: Example of Multiple Graph Panels on a Single Canvas



Graph Types

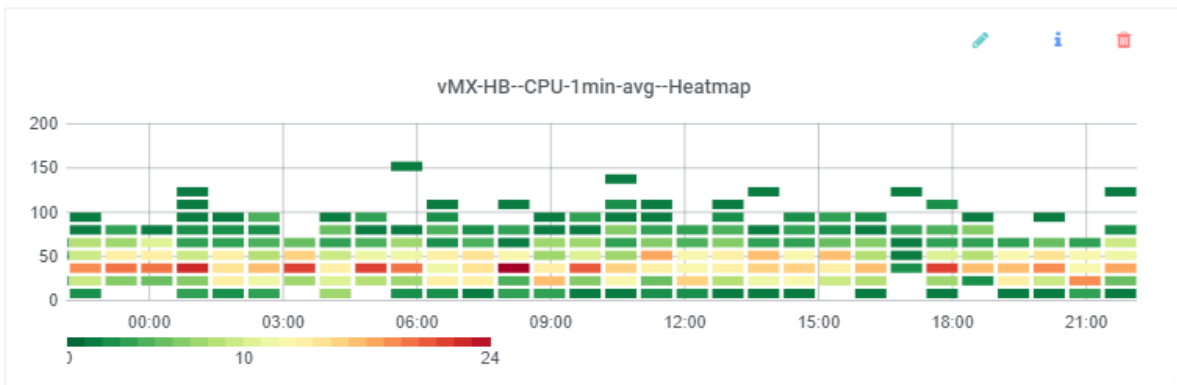
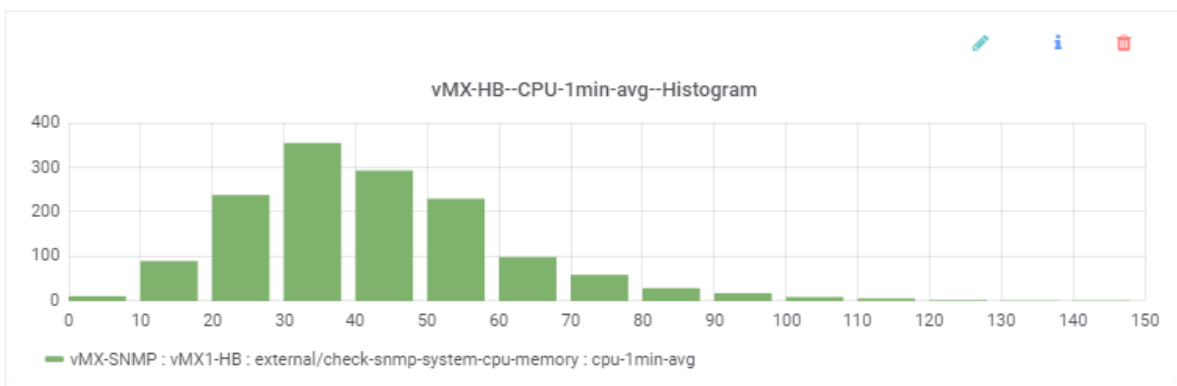
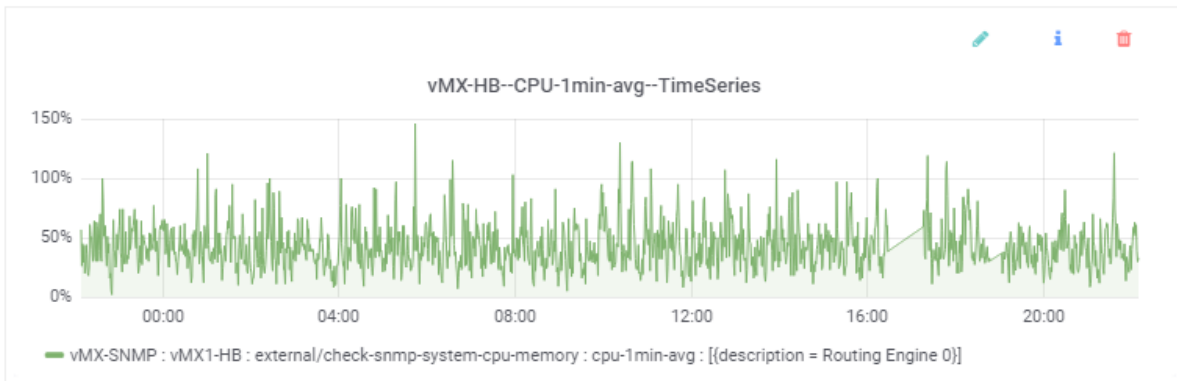
Graph types include time series graphs, histograms, and heatmaps.

Time series graphs are the kind you are used to, showing the data in a '2D' format where the x-axis indicates time while the y-axis indicates the value. Time series graphs are useful for real-time monitoring, and also to show historical patterns or trends. This graph type does not provide insight into whether a given value is 'good' or 'bad', it simply reports 'the latest value'.

Histograms work quite differently. Rather than show a continuous stream of data based on when each value occurred, histograms aggregate the data to show the *distribution* of the values over time. This results in a graph that shows 'how many instances of each value'. Histograms also show data in a '2D' format, however in this case the x-axis indicates the value while the y-axis indicates the number of instances of the given value.

Heatmaps bring together the elements above and provide a '3D' view to help determine the deviations in the data. Like a time series graph, the x-axis indicates time, while the y-axis indicates the value. Then the 'how many' aspect of a histogram is added in. Finally, the third dimension—*color*—is added. It is common to think of the colors as showing health, i.e., red means 'bad', yellow means 'OK', and green means 'good'. However, this is not correct; the color adds context. For each column, the bars indicate the various values that occurred. The color then indicates how often the values occurred relative to the neighboring values. Within each vertical set of bars, the values that occurred more frequently show as 'hotter' with orange and red, while those values that occurred less frequently show as shades of green.

To help illustrate these graph types, consider the graphs shown below.



All three graphs are showing the same data—the running 1-minute average of CPU utilization on a device over the last 24 hours. However, the way they visualize the data varies:

- The time series graph provides the typical view; each minute it adds the latest data point to the end of the line graph. Time moves forward along the x-axis from left to right, and the data values are indicated on the y-axis. What this graph doesn't show is how often each data point has occurred.
- The histogram groups together the values to show *how many* of each data point there are. Notice the tallest bar is the one between 30 and 40, which means the most common 1-minute CPU average value is in the 30-40% range. And how many times did this range of values occur? Based on the y-axis, there have been over 350 instances of values in this range. The next most frequently occurring values are in

the 40-50% range (almost 300 occurrences), while the 0-10% range has almost no occurrences, suggesting this CPU is rarely idle. What this graph doesn't show is how many of each data point occurred within a given time range.

- The heatmap makes use of elements from the other two graph types. Each small bar indicates that some number of instances occurred within the value range shown in the y-axis, at the given time shown in the x-axis. The color indicates which value ranges, for each given time, occurred *more* than others. To illustrate this, notice the vertical set of bars towards the right of the graph, at 18:00. In this example (at this zoom level), each column of vertical bars represents 12 minutes, and each small bar represents a bucket of 15 values. So the first (lowest) bar indicates that within this time range there were some values in the 0-14 range. The bar above indicates that within this time range there were some values in the 15-29 range, and so on. The color then indicates which bars have more values than others. In this example, the third bar is red indicating that for those 12 minutes most of the values fell into the 30-44 range (in this example the count is 21). By contrast, the first bar is the most green indicating that for those 12 minutes the least number of values fell into the 0-14 range (in this example the count is 1). This 'heat' information is also supported by the histogram; the most frequently occurring values were those in the 30-40 range, which indeed is the 'hotter' range in the heatmap.

How to Create Graphs

The configuration model for graphs is to create graph panels and group them into one or more canvases.

To create a new graph panel on a canvas:

1. Click the **Monitor > Graph** option in the left-nav bar.
2. Choose one of the following two options:
 - To create a graph in a new canvas, click the **+ New Canvas** button.
 - To create a graph within an existing canvas, select the desired canvas in the Saved Canvas and then click the **Add Graph +** button.
3. In the General section, provide the general descriptive information for the canvas (new canvas only) and graph:

Attribute	Description
Canvas Name	Name of the canvas
Description	(Optional) Description for the canvas.
Graph Name	Name of the graph panel.
Graph Type	Options include Time Series , Histogram , and Heatmap .

Attribute	Description
Time Range	<p>Choose the time range for the graph.</p> <p>In real-time graphs, the time range sets the x-axis range. For example, selecting 12 hrs means the x-axis shows the last 12 hours of data.</p>

4. Move down to the **Query** section. In the **FROM** section, define from where the data for the graph is coming:

Attribute	Description
Group	Choose the device group or network group.
Device	Choose a device from the group.
Topic / Rule	Choose the HealthBot topic/rule name.

5. In the **SELECT** section, select the data field and apply aggregation and transformation types to the data:

Attribute	Description
Field	<p>Choose a field name.</p> <p>This list is derived based on the fields defined in the selected topic/rule.</p>
Aggregation	(Optional) In the drop-down list, choose a data aggregation type.
Transformation	(Optional) In the drop-down list, choose a data transformation type.

6. In the **WHERE** section, filter data based on field and KPI key:

Attribute	Description
Tag Key / Field	<p>(Optional) Choose a key or field. A key is an index field such as interface name.</p> <p>This list is derived based on the keys and fields defined in the selected topic/rule.</p>

7. In the **GROUP BY** section, specify how to group the data based on time interval, fill, and KPI keys:

Attribute	Description
\$_interval	(Optional) Specify a time interval by which to group the data.
fill(null)	(Optional) Choose how to show a time interval when no data arrives from the device: null —(default) Report the timestamp and null as the output value. none —Report no timestamp and no output value. 0 —Report 0 as the output value. previous —Report the value from the previous time interval as the output value. linear —Report the results of linear interpolation as the output value.
Tag Key (the + icon)	(Optional) Choose a tag by which to group the data. A key is an index field such as interface name. This list is derived based on the keys and fields defined in the selected topic/rule.

8. (Optional) Move down to the **Visualization** section, and define y-axis details.

9. Click **Save** to save the graph and display the data.

Managing Graphs

- To edit a graph, click the pencil icon located in the top right corner of the graph itself.
- To delete a graph, click the trash can icon located in the top right corner of the graph itself.
- To delete a canvas, click the trash can icon located in the top right corner of the canvas.

Graph Tips and Tricks

- To sort canvases on the Saved Canvas page, click on the column headings.
- To reorganize graphs on the screen, hover your mouse near the upper-left corner of a graph panel and click-and-drag it to the desired position.
- To resize a graph, hover your mouse over the lower-right corner of the graph panel and click-and-drag it to the desired size.
- To change the color of graph elements, click the color bar for the desired line item under the graph.

- To zoom in on a graph, click and drag across the desired section of the graph; to zoom out, double-click on the graph.
- To isolate an element on the graph, click its related line item under the graph; to view all elements again, click the same line item.

Use Cases

How do I monitor interface flaps for a single interface?

Playbook used: interface-kpis-playbook

Graph configuration

Edit Graph

General

Canvas Name* Description

Graph Name* Time Series 3 Hour

Query

FROM

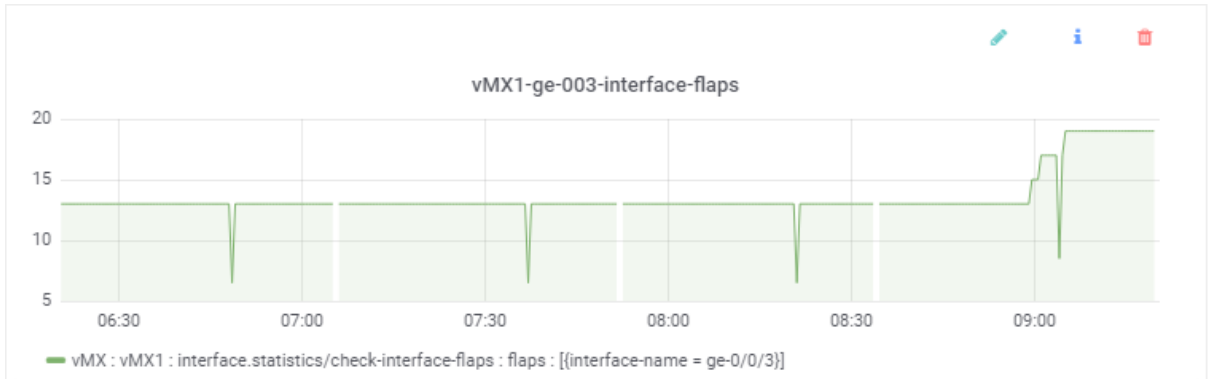
SELECT

WHERE =

GROUP BY

Visualization

Graph panel



How do I monitor interface flaps for all 'ge' interfaces on a device in a single graph?

Playbook used: interface-kpis-playbook

Graph configuration

Edit Graph

General

Canvas Name* TEST-canvas3 Description

Graph Name* vMX--interface-flaps Time Series 3 Hour

Query

FROM vMX vMX1 interface.statistics/check-interface-flaps

SELECT flaps mean() Transformation

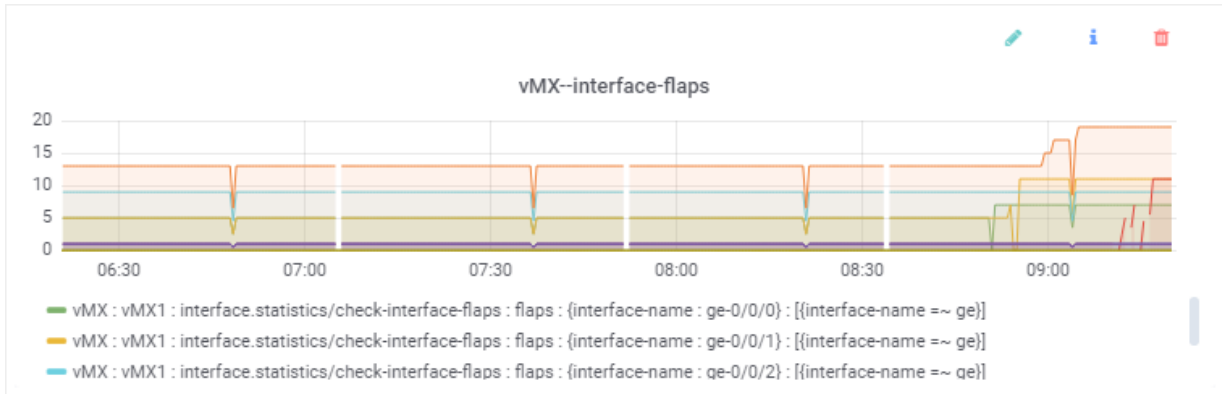
WHERE interface-name =~ (matches with) ge AND

GROUP BY \$__interval fill(null) interface-name +

Visualization

Cancel Save

Graph panel



How do I monitor system memory usage for all devices in a device group in a single graph?

Playbook used: system-kpis-playbook

Graph configuration

Edit Graph

▼ General

Canvas Name*	Canvas1	Description	
Graph Name*	All-devices--system-memory	Time Series	6 Hour

▼ Query

FROM	vMX	vMX1	system.memory/check-system-memory
SELECT	re-memory-buffer	mean()	Transformation
WHERE	+		
GROUP BY	1m	fill(null)	+

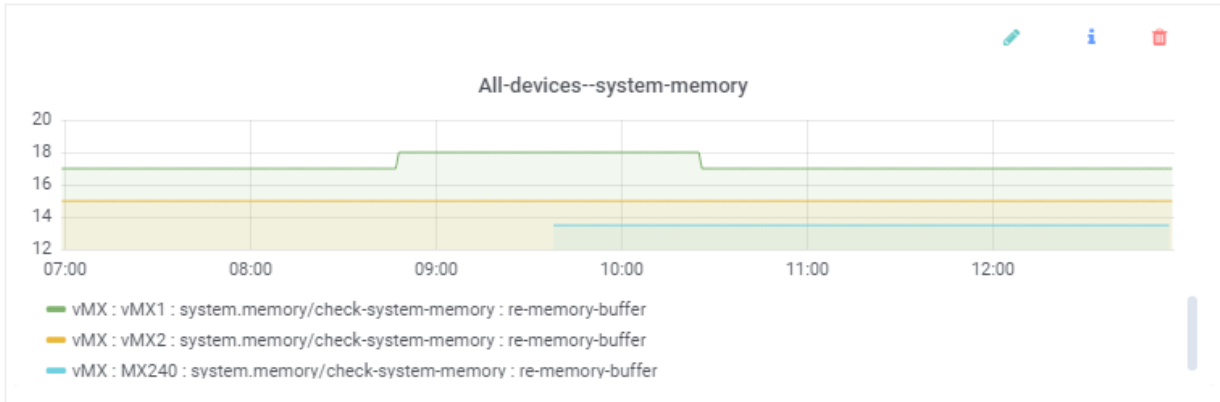
FROM	vMX	vMX2	system.memory/check-system-me...
SELECT	re-memory-buffer	mean()	Transformation
WHERE	+		
GROUP BY	1m	fill(null)	+

FROM	vMX	MX240	system.memory/check-system-me...
SELECT	re-memory-buffer	mean()	Transformation
WHERE	+		
GROUP BY	\$_interval	fill(null)	+

► Visualization

Cancel Save

Graph panel

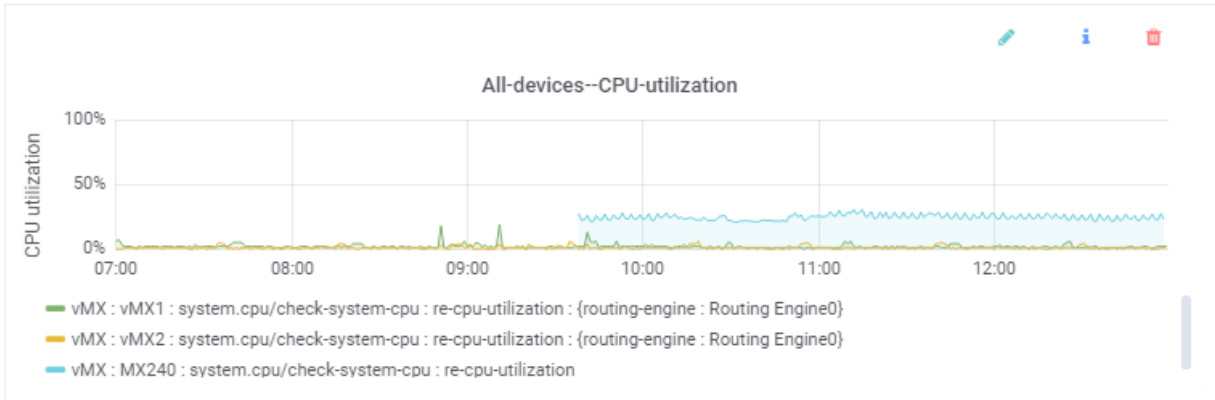


How do I monitor RE CPU usage for multiple devices in a single graph?

Playbook used: system-kpis-playbook

Graph configuration

Graph panel



How do I monitor RE CPU usage for multiple devices side by side?

Playbook used: system-kpis-playbook

Graph configuration

Add Graph

General

Canvas Name* Description

Graph Name* Time Series 6 Hour

Query

FROM

SELECT

WHERE

GROUP BY

Visualization

Edit Graph

General

Canvas Name* Description

Graph Name* Time Series

Query

FROM

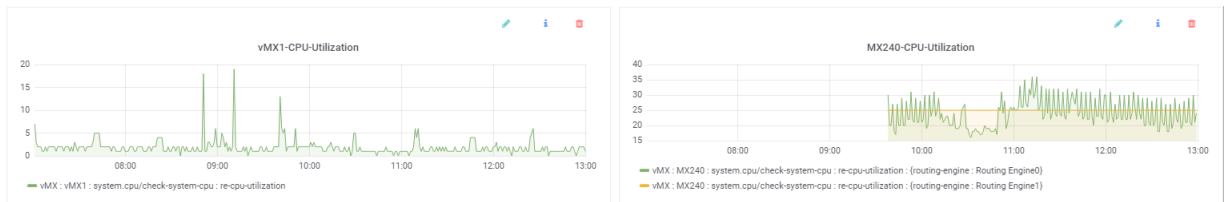
SELECT

WHERE

GROUP BY

Visualization

Graph panel



Release History Table

Release	Description
3.2.0	Starting with HealthBot Release 3.2.0, a new view called Time Inspector is available

RELATED DOCUMENTATION

Alarms and Notifications

IN THIS SECTION

- [Generate Alarm Notifications | 181](#)
- [Manage Alarms Using Alarm Manager | 190](#)
- [Stream Sensor and Field Data from HealthBot | 195](#)

Generate Alarm Notifications

HealthBot generates alarms that indicate when specific KPI events occur on your devices. To receive HealthBot notifications for these KPI events, you must first configure a notification profile. Once configured, you can enable alarm notifications for specific device groups and network groups.

HealthBot supports the following notification delivery methods:

- Web Hook
- Slack
- Kafka Publish
- Microsoft Teams (HealthBot 2.1.0 and later)
- Email (HealthBot 2.1.0 and later)

This section includes the following procedures:

- [Configure a Notification Profile on page 182](#)
- [Enable Alarm Notifications for a Device Group or Network Group on page 190](#)

Configure a Notification Profile

A notification profile defines the delivery method to use for sending notifications.

1. Click the **Settings > System** option in the left-nav bar.
2. Click the **Notification** tab on the left of the window. click the add notification button (**+ Notification**).
3. Click the **+ Notification** button
4. In the **Add Notification** window that appears, configure the notification profile:

-

Attributes	Description
Name	Enter a name.
Description	(Optional) Enter a description.
Notification Type	Select a notification type: <ul style="list-style-type: none"> • Web Hook • Slack • Kafka Publish • Microsoft Teams (HeathBot 2.1.0 and later) • EMails (HeathBot 2.1.0 and later) Notification type attributes vary based on notification type selected. See below for details.

5. Click **Save and Deploy**.

NOTIFICATION TYPE DETAILS

Web Hook

- **URL**—URL at which the Web Hook notification should be posted.
- **Username**—(Optional) Username for basic HTTP authentication.
- **Password**—(Optional) Password for basic HTTP authentication.

Slack

- **URL**—URL at which the Slack notification should be posted. Different from your Slack workspace URL. Go to <https://slack.com/services/new/incoming-webhook> and sign in to your Slack workspace to create a Slack API endpoint URL.
- **Channel**—Channel on which the notification should be posted.

Kafka Publish

- **Bootstrap Servers**—Add Kafka host:port pairs from the drop-down list to establish the initial connection to the Kafka cluster.
- **Topic**—(Optional) Name of the Kafka topic to which data will be published. By default, the Kafka topic naming convention for device group alarm notifications is *device-group.device-id.topic.rule.trigger*.

Depending on the authentication protocols being used, the required authentication parameters are as follows:

Protocol	Required Parameters
SASL/SSL	Username, password and certificate
SASL/Plaintext	Username and password
SSL	Certificate
Plaintext	None

- **Username**—Username for SASL/SSL or SASL/plaintext authentication.
- **Password**—Password for SASL/SSL or SASL/plaintext authentication.
- **Certificate**—Kafka server's CA certificate. Choose file from the drop-down list.
- **Upload Certificate**—Location from where the Kafka server's CA certificate will be uploaded. Click **Choose files** and navigate to the file location. File should be in Privacy Enhanced Mail (PEM) format.

Microsoft Teams

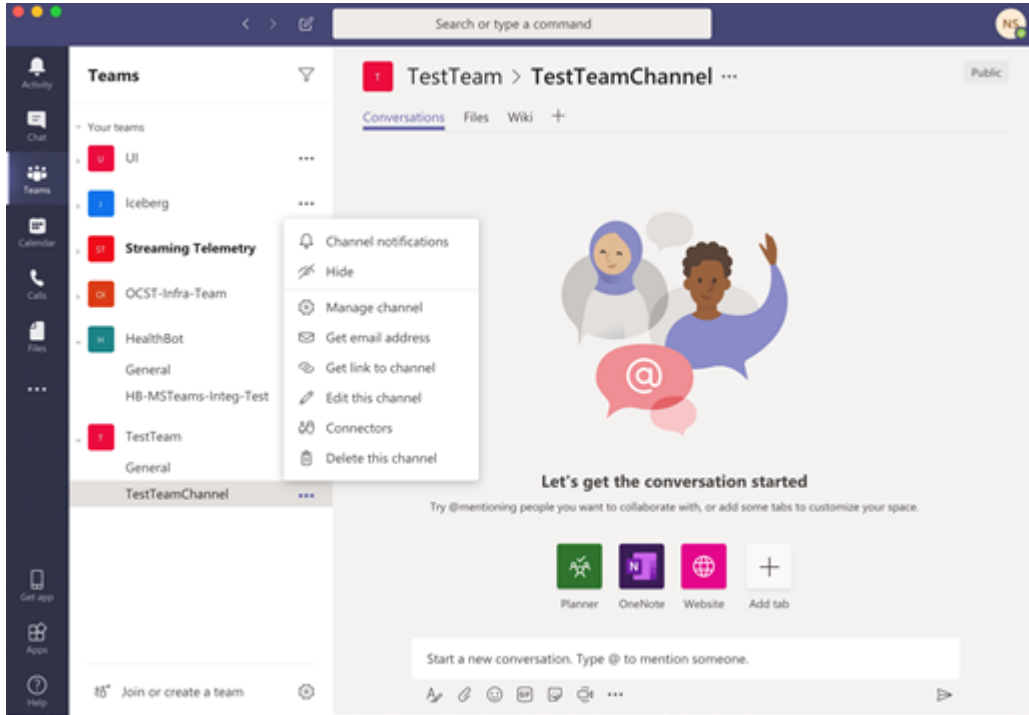
As of HealthBot 2.1.0, you can send HealthBot notifications to Microsoft Teams. Teams can provide a connector which you can add to HealthBot to enable the connection.

Configuration workflow:

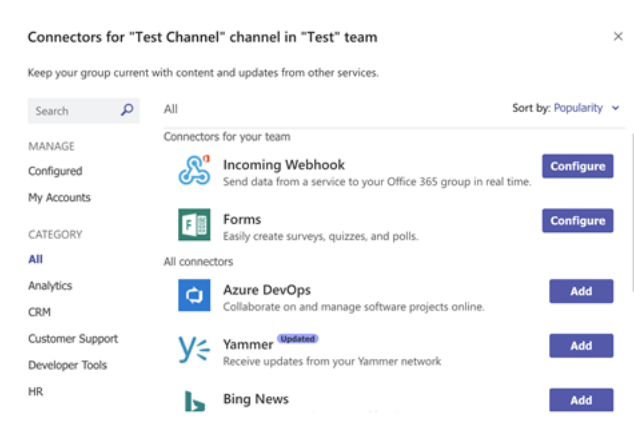
- In Teams, create a new connector set as an incoming webhook.
- Copy the URL provided by Teams.
- In HealthBot, configure a notification profile that sends to Microsoft Teams.
- Apply the notification profile to a device group.

To configure MS Teams notifications:

1. In Teams, select the desired channel and click the ellipsis (...).
2. In the menu that appears, click **Connectors**.



3. Use the Incoming Webhook option and click **Configure**.

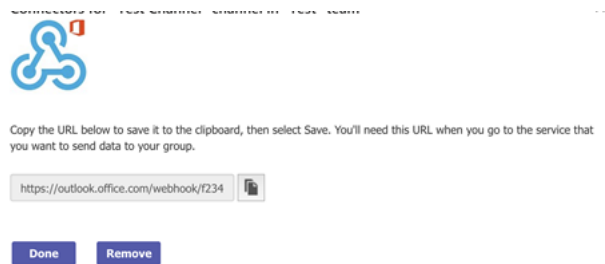


4. On the next page, click **Create**.

Customize the image to associate with the data from this Incoming Webhook.



- Once the web hook is successfully created, copy the provided URL.



- In HealthBot, go to the **Settings > System** page select the Notification tab.
- Click the **+ Notification** button.
- Configure the notification profile as follows:
 - Name - Enter a profile name.
 - Notification Type - select **Microsoft Teams**.
 - Channel - Paste the URL provided by the Teams UI above.
- Click **Save and Deploy**.
- Apply the notification profile to a device group or network group as shown in [“Enable Alarm Notifications for a Device Group or Network Group” on page 190](#)

EMails

As of HealthBot 2.1.0, you can send HealthBot notifications by email. By default, email notifications cover all running playbooks and rules for the device group or network group to which they are applied, however you can narrow the focus by selecting specific rules.

NOTE: HealthBot includes its own mail transfer agent (MTA), so no other mail server is required.

Configuration workflow:

- In HealthBot, configure a notification profile that sends to email.
- Apply the notification profile to a device group.

To configure email notifications:

1. In HealthBot, go to the **Settings > System** page.
2. Select the **Notification** tab and click the the **+ Notification** button.
3. Configure the notification profile as follows:
 - Name - Enter a profile name.
 - Notification Type - Select **Emails**.
 - Email Addresses - Enter an email address and click **Add <email-address>**; repeat for more email addresses.
 - (Optional) Rule filters - To narrow the scope of what triggers an email, define rule filters. Enter a filter and click **Add <rule-filters>**; repeat for more filters.
 - Format is topic/rule; can use regular expressions
 - Example: **interface.statistics/check-interface-flaps** sends notifications only for the rule check-interface-flaps.
 - Example: **system.processes/.*** , **system.cpu/.*** , and **interface.statistics/.*** sends notifications for all rules under the topics system.processes, system.cpu, and interface.statistics.
4. Click **Save and Deploy**.
5. Apply the notification profile to a device group or network group as shown in [“Enable Alarm Notifications for a Device Group or Network Group” on page 190](#)



Enable Alarm Notifications for a Device Group or Network Group

To enable alarm notifications for a device group or network group:

1. For Device Groups, select the **Configuration > Device Group** page from the left-nav bar.
For Network Groups, select the **Configuration > Network** page from the left-nav bar.
2. Click the name of the device group or network group for which you want to enable alarm notifications.
3. Click the **Edit (Pencil)** icon.
4. Scroll down to the **Notification** section in the pop-up window and click the caret to expand that section.
5. Select a destination for any alarm level (Major, Minor, or Normal) that you want. Notification can be sent to zero or more defined destinations for each alarm level.
6. Click **Save and Deploy**.

Manage Alarms Using Alarm Manager

You can use the Alarm Manager feature to organize, track, and manage KPI event alarm notifications received from HealthBot devices. The Alarm Manager does not track alarms by default; it is populated based on which device groups or network groups are configured to send the notifications.

Viewing Alarms

To view the alarms report table, go to the **Monitor > Alarms** page in the left-nav bar. Note that Alarm Manager consolidates duplicate alarms into one table entry and provides a count of the number of duplicate alarms it has received.

The following table describes the alarms report table attributes.

Attributes	Description
Severity	Severity level of the alarm. Options include: <ul style="list-style-type: none"> • Major • Minor • Normal
Status	Management status of the alarm entry. Options are Open, Active, Shelved, Closed, and Ack. The statuses available in the Status pull-down menu in the top row of the table only include statuses of alarms visible in the table and those allowed by the status filter above the table.
Last Received	Time the alarm was last received.
Dupl.	Duplicate count. Number of times an alarm with the same event, resource, environment, and severity has been triggered.
Topic	Device component topic name.
Resource	Device name.
Event	Name of the rule, trigger or field, and event with which the alarm is associated.
Text	Health status message.

The following table describes the main features of the alarms report table:

Feature	Description
Sort the data by ascending or descending order based on a specific attribute.	Click on the name of the data type at the top of the column by which you want to sort.
Filter the data based on the device group.	In the drop-down list at the top left corner of the page, select a device group by which to filter.

Feature	Description
Filter the data based on the alarm status.	<p>Two options:</p> <ol style="list-style-type: none"> 1. In the drop-down list above the table at the top of the page, select one or more status types on which to filter. Options are open, active, shelved, closed, and ack. You can filter on multiple status types. 2. In the drop-down list at the top of the Status column, select a status type by which to filter. Note that if there are status types shown in the filter list at the top of the report, then the status column can only show those status types.
Filter the data based on the severity, topic, or resource	In the associated drop-down list for each attribute at the top of the table, select an option by which to filter.
Filter the data based on a keyword.	In the associated text box under the Event or Text attribute name at the top of the table, enter the keyword on which to filter.
Filter the data based on date or time received.	In the Last Received field, enter a date and time in the format: <Day> <DD> <Mon> <HH:MM>
Navigate to a different page of the table.	<p>Two options:</p> <ol style="list-style-type: none"> 1. At the bottom of the table, click the Previous or Next buttons. 2. At the bottom of the table, select the page number using the up/down arrows (or by manually entering the number) and then press Enter.
Change the number of rows displayed.	At the bottom of the table, choose the number of rows to display in the drop-down list. The table displays 20 rows by default.
If the data in a cell is truncated, view all of the data in a cell.	Resize the column width of the cell by dragging the right side of the title cell of the column to the right.

Manage Individual Alarms

You can view detailed information about each alarm in the alarms report table. You can also assign a management status (such as open, ack, and close), and apply simple actions (such as shelve and delete) to each alarm.

To manage individual alarms:

1. Go to the **Monitor > Alarms** page from the left-nav bar to open the alarms report table.
2. Click on a single alarm entry in the table. The Alarm Details pane displays detailed information about the alarm.

The following table describes the set of buttons at the top of the Alarm Details pane:

Button	Description
Open	Changes the status of the alarm to Open.
Shelve	Removes the alarm from the table for a set amount of time. Time options are 1, 2, 4 and 8 hours. Click Unshelve to disable this feature.
Ack	Changes the status of the alarm to Ack. The Ack status removes the alarm from the table, but the alarm still remains active.
Close	Changes the status of the alarm to Closed. The Closed status indicates that the severity level of the alarm is now Normal.
Delete	Deletes the alarm from the table.

Configure Alarm Blackouts

You can configure blackout periods to suppress or mute alarms during, for example, scheduled downtimes.

To configure blackouts:

1. Click the **Settings > System** page from the left-nav bar.
2. Select the **Alarm** tab on the left side of the page.
3. In **Alarm Blackout Settings**, click the **+ Alarm Blackout** button.
4. Enter the necessary values in the text boxes for the blackout configuration.

The following table describes the attributes in the **Add an Alarm Blackout** pane:

Attributes	Description
Duration	Select a start and end date and time for the blackout.
Device Group	Select a device group from the drop-down list to which to apply the blackout configuration.
Attribute	(Optional) Specify an attribute from the drop-down list to which to apply the blackout configuration.
Value	(Optional) If a blackout attribute is specified, provide an associated value (as shown in the alarms report table). Only the alarms that match this attribute value exactly will be suppressed from the alarms report table. NOTE: For the Resource-Event attribute, you must specify a resource from the drop-down list, as well as specify an Event value. Only the alarms generated by the specified resource that match this Event value exactly will be suppressed from the alarms report table.

5. Click **Save** to save the configuration.
6. (Optional) Click the **Delete** button to delete a blackout configuration.

Stream Sensor and Field Data from HealthBot

You can configure HealthBot to publish HealthBot sensor and field data for a specific device group or network group. You must first configure the notification type for publishing and then specify the fields and sensors that you want published.

Configure the Notification Type for Publishing

HealthBot supports Apache Kafka for publishing sensor and field data. You must first configure a Kafka publishing profile before you can start publishing sensor and field data for a specific device group or network group.

To configure a Kafka publishing profile:

1. Select the **Settings > System** page from the left-nav bar.
2. Click the **Notification** tab on the left part of the page.
3. In **Notification Settings**, click the **+ Notification** button.
4. Enter the necessary values in the text boxes and select the appropriate options for the Kafka publishing profile.

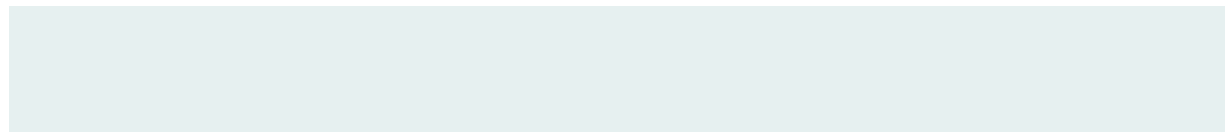
The following table describes the relevant attributes in the **Add a Notification Setting** and **Edit Notification Configuration** panes:

Attributes	Description
Name	Name of the notification.
Description	(Optional) Description of the notification.
Notification Type	Click the Kafka publish radio button.

Attributes	Description										
Kafka Publish	<ul style="list-style-type: none"> • Bootstrap Servers—Add Kafka host:port pairs from the drop-down list to establish the initial connection to the Kafka cluster. • Topic—(Optional) Name of the Kafka topic to which data will be published. By default, the Kafka topic naming conventions are: <ul style="list-style-type: none"> • For device group field data, <i>device-group.device-id.topic.rule.fields</i> • For network group field data, <i>network-group.topic.rule.fields</i> • For device group sensor data, <i>device-group.device-id.sensors</i> <p>Depending on the authentication protocols being used, the required authentication parameters are as follows:</p> <table border="1" data-bbox="428 709 1607 1087"> <thead> <tr> <th>Protocol</th> <th>Required Parameters</th> </tr> </thead> <tbody> <tr> <td>SASL/SSL</td> <td>Username, password and certificate</td> </tr> <tr> <td>SASL/Plaintext</td> <td>Username and password</td> </tr> <tr> <td>SSL</td> <td>Certificate</td> </tr> <tr> <td>Plaintext</td> <td>None</td> </tr> </tbody> </table> <ul style="list-style-type: none"> • Username—Username for SASL/SSL or SASL/plaintext authentication. • Password—Password for SASL/SSL or SASL/plaintext authentication. • Certificate—Kafka server’s CA certificate. Choose file from the drop-down list. • Upload Certificate—Location from where the Kafka server’s CA certificate will be uploaded. Click Choose files and navigate to the file location. File should be in Privacy Enhanced Mail (PEM) format. 	Protocol	Required Parameters	SASL/SSL	Username, password and certificate	SASL/Plaintext	Username and password	SSL	Certificate	Plaintext	None
Protocol	Required Parameters										
SASL/SSL	Username, password and certificate										
SASL/Plaintext	Username and password										
SSL	Certificate										
Plaintext	None										

5. Click **Save** to save the configuration or click **Save and Deploy** to save and deploy the configuration.

6. Apply the Kafka publishing profile to a device group or network group. For more details, see the [“Publish Data for a Device Group or Network Group” on page 199](#) section.



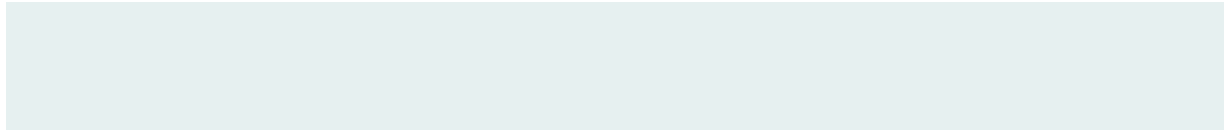
Publish Data for a Device Group or Network Group

To publish HealthBot sensor or field data for a device group or network group:

1. Select the **Configuration > Device Group** page from the left-nav bar.
2. Click the name of the device group to which you want to publish data.
3. Click the **Edit Device Group (Pencil)** icon.
4. Under **Publish**, select the appropriate Destinations, Field, or Sensor from the drop-down lists for the data you want to publish. To publish field or sensor data, you must configure a destination.

Parameter	Description
Destinations	<p>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 option in the left-nav bar. The publishing profiles are listed under Notification Settings.</p> <p>NOTE: Only Kafka publishing is currently supported.</p>
Field	Select the HealthBot rule topic and rule name pairs that contain the field data you want to publish.
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.

5. Click **Save** to save the configuration or click **Save and Deploy** to save and deploy the configuration.



Release History Table

Release	Description
2.1.0	As of HealthBot 2.1.0, you can send HealthBot notifications by email.

RELATED DOCUMENTATION

| [Monitor Device and Network Health](#) | 148

Generate Reports

You can generate HealthBot reports for device groups and network groups. These reports include alarm statistics, device or network health data, as well as device-specific information (such as hardware and software specifications).

HealthBot's reporting functionality allows you to:

- Send reports by email, save them on the HealthBot server, or download them to your local machine
- Schedule reports to run at regular intervals, or for a specific time
- Generate reports on demand (HealthBot 2.1.0 and later)
- Compare (diff) two reports (HealthBot 2.1.0 and later)
- Capture a snapshot of a specific set of fields at a given point in time (HealthBot 3.1.0 and later)

This section includes the following procedures:

- [Generate On-Demand Reports on page 201](#)
- [Generate Scheduled Reports on page 204](#)
- [View Reports on page 208](#)
- [Create Field Snapshot on page 211](#)
- [Compare \(Diff\) Reports on page 213](#)

Generate On-Demand Reports

You can generate and download a report on demand for a device group or network group. As with regular report generation, formats supported include HTML or JSON, and you can download the report or receive it by email.

Once generated, you can re-download on-demand reports from the Reports page. These reports have the report name **HB_MANUAL_REPORT**.

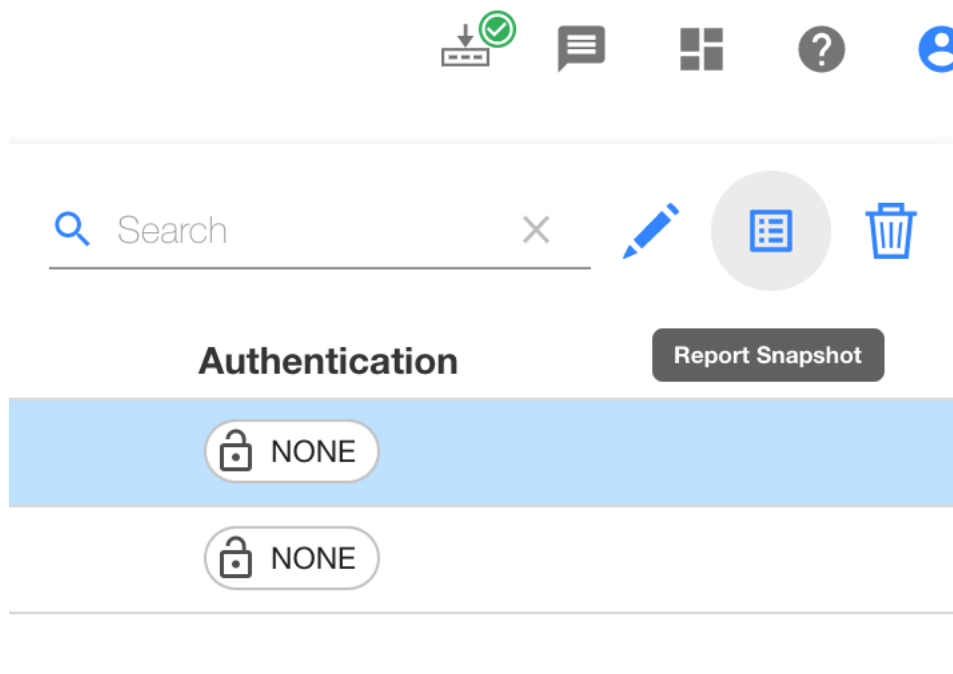
To generate an on-demand report:

1. For a device group, navigate to the **Configuration > Device Group** page from the left-nav bar and select the device group name from the list.

For a network group, navigate to the **Configuration > Network** page from the left-nav bar and select the network group name from the list.

2. Click the **Report Snapshot (Page)** icon in the upper right part of the page as shown in

Figure 53: Report Snapshot Button



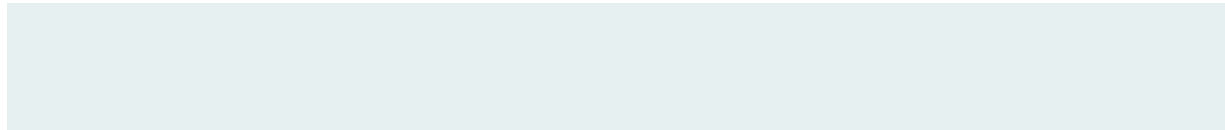
3. On the page that appears, enter report generation details, including:

- Format - **HTML** or **JSON**
- Destination Type - save to your computer (**disk**) or send to **email** (which also saves to disk)
 - If disk, specify the number of reports to save on the server before deleting older reports

- If email, add target email address(es)
- (Optional) Select graph canvases to include in the report.
- (Optional) Select the desired graph panels to include in the report.

4. Click **Submit**.

5. A dialog box appears allowing you to download the file. Additionally, if the destination is disk, HealthBot stores a copy of the report on the server. If the destination is email, HealthBot sends the report to the specified account.



Generate Scheduled Reports

The workflow to configure and generate scheduled reports is as follows:



Create a Destination Profile

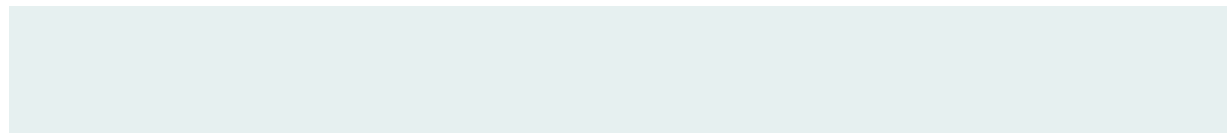
1. Select the **Settings > System** page from the left-nav bar.
2. Click the **Destination** tab on the left of the page.
3. In the **Destination Settings** section, click the **+ Destination** button.
4. Specify the destination profile settings as appropriate.

The following table describes the attributes in the **Add a destination** and **Edit a destination** panes:

Attributes	Description
Destination Name	Enter a name. The name cannot be changed once saved.
Destination Type	Options include Email or Disk .
Email > Email Id	Enter an email address to which the report will be sent.
Disk > Maximum Reports	Specify how many versions of this report will be stored on the server. Older reports are deleted as newer reports are generated and saved.

NOTE: Using the email option also saves a copy of the report to disk.

5. Click **Save and Deploy**.



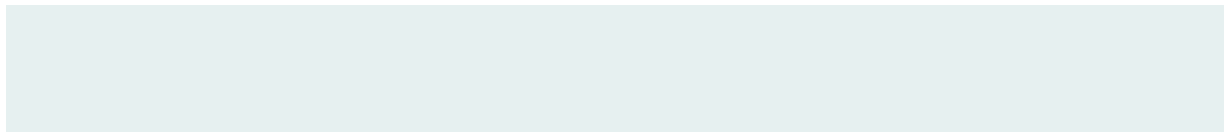
Create a Schedule Profile

1. Click the **Scheduler** tab on the left side of the page.
2. In the **Scheduler Settings** section, click the **+ Scheduler** button.
3. Specify the schedule profile settings as appropriate.

The following table describes the attributes in the **Add a scheduler** and **Edit a scheduler** panes:

Attributes	Description
Name	Enter a name .
Scheduler Type	Select continuous .
Start On	Select the date and time for the first report to be generated.
Run for	Not applicable.
End On	(Optional) Select the date and time to stop generating reports. Leave blank to generate the report indefinitely.
Repeat	Select one of the following: <ul style="list-style-type: none"> • The frequency (every day, week, month, or year) at which you want the report to be generated. • Never—generate the report only once. • Custom—select and use the Repeat Every fields to configure a custom frequency.

4. Click **Save and Deploy**.



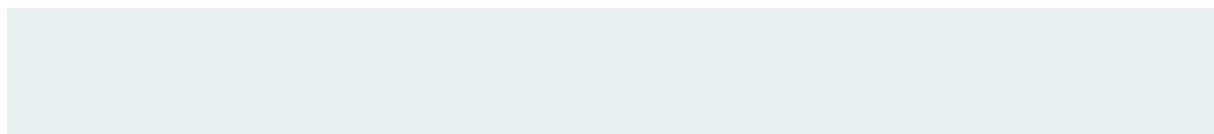
Create a Report

1. Click the **Report** tab on the left side of the page.
2. In the **Report Settings** section, click **+ Report** button.
3. Specify the schedule profile settings as appropriate.

The following table describes the attributes in the **Add a report setting** and **Edit a report setting** panes:

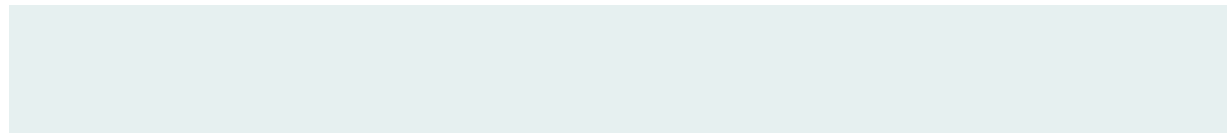
Attributes	Description
Name	Enter a name.
Format	Options include HTML and JSON .
Schedule(s)	Select the schedule profile you created above.
Destination(s)	Select the destination profile you created above.
Canvas(es)	(Optional) Select graph canvases to include in the report. The list of graph panels in the Panel(s) drop-down list changes based on the canvas selected. For information on creating graphs, see Graph Page .
Panel(s)	(Optional) Select the desired graph panels to include in the report. NOTE: JSON reports include the raw time series data only, no graphs.

4. Click **Save and Deploy**.



Associate the Report to a Device Group or Network Group

1. For a device group, select the **Configuration > Device Groups** page from the left-nav bar.
For a network group, select the **Configuration > Network** page from the left-nav bar.
2. Click the name of the device group or network group for which you want to generate reports.
3. Click the **Edit (Pencil)** button.
4. In the **Reports** section, click the caret to expand the menu.
5. Click the name(s) of the reports that you want to associate with this group.
6. Click **Save and Deploy**.



View Reports

To view reports for a device group or network group:

1. If the report's notification parameters are set to use email, check the email box of the specified account for the report and open the attachment.
2. If the report's notification parameters are set to save to the server's disk (or even if set to email), select the **Monitor > Reports** page from the left-nav bar. The reports are organized by the date and time at which they were generated. The most recent report is listed at the top of the table.
3. Find the report you wish to download. To help find the desired report:
 - Click the column headings to sort based on that column.
 - Search within a column using the text box under the column heading.
 - Use the bottom of the page to view more rows or change pages.
4. Click on the name of the report to download it to your system.

Sample Report

A report shows the following information:

Healthbot Report for Device group "vSRX"

Report Name: "TEST-sched-report-TEST-report-vSRX"

Generation Time: "2019-11-19T19:10"

- [Report 1_Dashboard](#)
- [Report 2_Device State](#)
- [Report 3_Alert State](#)
- [Report 4_Device Inventory](#)

Dashboard

Device State view

Total : 1 Devices
 Risk : 0 Devices
 Check : 1 Devices
 No Data : 0 Devices
 Good : 0 Devices



[Top of page](#)

Alert State View

Total : 27
 Critical : 0
 Major : 26
 Minor : 1
 Normal : 0

Critical 0%
 Major 96%
 Minor 4%
 Normal 0%

Alerts of top 5 devices

vSRX : 27
 vSRX 100%

Alerts of top 5 topics

system.cpu/check-system.cpu-load-average : 4
 interface.statistics/check-interface-status : 2
 system.memory/check-system-memory : 2
 interface.statistics/check-neighbor-state : 2
 interface.statistics/check-out-traffic : 2

system.cpu/check-system.cpu-load-average 33%
 interface.statistics/check-interface-status 17%
 system.memory/check-system-memory 17%
 interface.statistics/check-neighbor-state 17%
 interface.statistics/check-out-traffic 17%

Device State

Total Devices Displayed: 1 Page 1 of 1

State	Device Name
yellow	vSRX

[Top of page](#)

Alert State

Total Alerts Displayed: 20 Page 1 of 2

Severity	Status	Last Receive Time	Duplicate Count	Environment	Topic	Resource	Event	Text
major	open	2019-11-19T12:55:28.062Z	249	Production.vSRX	interface.statistics	vSRX	Rule=interface.statistics/check-ifl-out-traffic, Sensor=interfaces/, event=no-data-received	No data is being received on Sensor
major	open	2019-11-19T12:55:28.801Z	249	Production.vSRX	interface.statistics	vSRX	Rule=interface.statistics/check-out-traffic, Sensor=interfaces/, event=no-data-received	No data is being received on Sensor
major	open	2019-11-19T12:56:58.367Z	124	Production.vSRX	system.cpu	vSRX	Rule=system.cpu/check-system.cpu-load-average, Sensor=CPUUtilizationTable, event=no-data-received	No data is being received on Sensor
major	open	2019-11-19T12:56:58.784Z	249	Production.vSRX	interface.statistics	vSRX	Rule=interface.statistics/check-interface-flaps, Sensor=interfaces/, event=no-data-received	No data is being received on Sensor
major	open	2019-11-19T12:55:28.767Z	249	Production.vSRX	interface.statistics	vSRX	Rule=interface.statistics/check-interface-status, Sensor=interfaces/, event=no-data-received	No data is being received on Sensor
major	open	2019-11-19T12:55:28.751Z	249	Production.vSRX	interface.statistics	vSRX	Rule=interface.statistics/check-neighbor-state, Sensor=interfaces/, event=no-data-received	No data is being received on Sensor
major	open	2019-11-19T12:55:28.681Z	249	Production.vSRX	interface.statistics	vSRX	Rule=interface.statistics/check-in-traffic, Sensor=interfaces/, event=no-data-received	No data is being received on Sensor
major	open	2019-11-19T12:56:58.796Z	124	Production.vSRX	system.cpu	vSRX	Rule=system.cpu/check-system.cpu, Sensor=components/, event=no-data-received	No data is being received on Sensor
major	open	2019-11-19T12:55:28.726Z	249	Production.vSRX	interface.statistics	vSRX	Rule=interface.statistics/check-ifl-in-traffic, Sensor=interfaces/, event=no-data-received	No data is being received on Sensor
major	open	2019-11-19T12:56:58.755Z	124	Production.vSRX	interface.statistics	vSRX	Rule=interface.statistics/check-in-errors, Sensor=interfaces/, event=no-data-received	No data is being received on Sensor
major	open	2019-11-19T12:55:28.710Z	249	Production.vSRX	interface.statistics	vSRX	Rule=interface.statistics/check-out-errors, Sensor=interfaces/, event=no-data-received	No data is being received on Sensor
major	open	2019-11-19T12:56:58.683Z	124	Production.vSRX	system.memory	vSRX	Rule=system.memory/check-system-memory, Sensor=components/, event=no-data-received	No data is being received on Sensor
major	open	2019-11-19T12:56:58.055Z	93	Production.vSRX	interface.statistics	vSRX	Rule=interface.statistics/check-in-errors, Trigger=in-errors, event=no-data-received	No data present for last 180 seconds
major	open	2019-11-19T12:56:57.060Z	93	Production.vSRX	system.cpu	vSRX	Rule=system.cpu/check-system.cpu, Trigger=re-cpu-utilization, event=no-data-received	No data present for last 300 seconds
major	open	2019-11-19T12:56:58.070Z	93	Production.vSRX	system.memory	vSRX	Rule=system.memory/check-system-memory, Trigger=re-memory-buffer-utilization, event=no-data-received	No data present for last 300 seconds
major	open	2019-11-19T12:56:58.067Z	93	Production.vSRX	interface.statistics	vSRX	Rule=interface.statistics/check-interface-flaps, Trigger=link-flaps, event=no-data-received	No data present for last 180 seconds
major	open	2019-11-19T12:56:57.060Z	93	Production.vSRX	interface.statistics	vSRX	Rule=interface.statistics/check-out-errors, Trigger=out-errors, event=no-data-received	No data present for last 180 seconds
major	open	2019-11-19T12:56:58.067Z	93	Production.vSRX	system.cpu	vSRX	Rule=system.cpu/check-system.cpu-load-average, Trigger=cpu-utilization-5min, event=no-data-received	No data present for last 120 seconds
major	open	2019-11-19T12:56:58.056Z	93	Production.vSRX	interface.statistics	vSRX	Rule=interface.statistics/check-ifl-out-traffic, Trigger=out-traffic, event=no-data-received	No data present for last 300 seconds
major	open	2019-11-19T12:56:58.063Z	93	Production.vSRX	interface.statistics	vSRX	Rule=interface.statistics/check-interface-status, Trigger=link-state, event=no-data-received	No data present for last 120 seconds

[Top of page](#)

Device Inventory

Device Id	Hostname	Serial Number	Product	Release	Platform
vSRX	SRX-PNF	1D530FD8888B		19.2R1.8	vSRX

[Top of page](#)

Plots

Create Field Snapshot

You can capture fields (and their values) from rules applied to deployed devices. In the HealthBot CLI, you identify the fields to capture by specifying an *xpath* as shown below (without spaces):

```
{
  capture-fields: [
    /device-group[device-group-name='DsGr1']/device[device-id='n20-1']/topic[topic-name='system.cpu']/rule[rule-name='check-system.cpu']/req.utilization,
    /device-group[device-group-name='DsGr2']/device[device-id='n20-1']/topic[topic-name='interface.statistics']/rule[rule-name='check-interfaces']/ineros.curt
  ]
}
```

In the HealthBot GUI, you specify the fields during the creation of a report. HealthBot takes care of creating the *xpath* from your configuration in the **Settings > System > Reports > Add a Report Setting** or the **Settings > System > Reports > Edit Report** window as shown in [Figure 54 on page 211](#) below:

Figure 54: Add/Edit Report Setting

Edit a Report Setting

Name*

Test-Report-1

Report Format*

HTML

Schedule(s)*

Hourly

Schedule(s) to run report

Destination(s)*

EmailAdmins

Canvas(es)

Panel(s)

Add Fields

DevGrp1

mx240-1

system.cpu/check-system-cpu

re-cpu-utilization Field(s)

interface.statistics/check-in-errors

in-errors-count Field(s)

Topic/Rule

Device

Device Group

CANCEL

SAVE

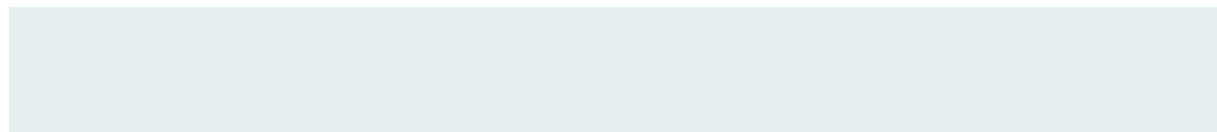
SAVE & DEPLOY

Compare (Diff) Reports

You can compare the differences between two reports for a device group or network group. The diff allows you to view added/removed/modified alerts, devices, health information, and graphs.

To generate a report diff:

1. On the Reports page, select two reports and click the **Diff Reports** button.
2. The diff opens as an HTML page in a new tab.



Sample Report Diff


A sample of a report diff for a device group is shown below.

DIFF

Header Diff

Field	Old	New
Name	S1-r1-dut_chk	S1-r2-dut_chk
Report	r1	r2
Time	2019-11-20T11:51:20	2019-11-20T11:28

Alert Data Diff



Added	Removed	Modified
0	5	6

Statistics Changes

Field	Old	New
SeverityCounts	Critical: 0 Major: 18 Minor: 1 Normal: 4	Critical: 0 Major: 18 Minor: 0 Normal: 0
StatusCounts	Ack: 0 Closed: 4 Open: 19 Shelved: 0 Expired: 0	Ack: 0 Closed: 0 Open: 18 Shelved: 0 Expired: 0
TotalAlerts	23	18
TotalMinorOpen	1	0
TotalNormalClosed	4	0

Added Alerts

None

Removed Alerts

Severity	Status	CreateTime	DuplicateCount	Environment	Service	Resource	Event	Text
minor	open	2019-11-20T11:53:20.688Z	0	Production.dut_chk	system.cpu	r0	Event	Routing Engine0 15min CPU utilization(71) is in medium range
normal	closed	2019-11-20T11:51:20.688Z	0	Production.dut_chk	system.cpu	r0	Event	Routing Engine0 5min CPU utilization(53) is normal
normal	closed	2019-11-20T11:49:20.692Z	0	Production.dut_chk	system.cpu	r0	Event	Routing Engine0 CPU utilization(1) is normal
normal	closed	2019-11-20T11:49:21.690Z	0	Production.dut_chk	system.processes	r0	Event	ppt_11_80000010 cpu utilization(0), is normal
normal	closed	2019-11-20T11:49:20.688Z	0	Production.dut_chk	system.cpu	r0	Event	Routing Engine0 1min CPU utilization(31) is normal

Modified Alerts

a0c2aa28-baf6-40c7-b2d8-c630c5713a1

Field	Old	New
duplicateCount	101	91

67789306-b131-46f8-b927-d99a0e0e3537

Field	Old	New
duplicateCount	101	91

88f32d5-3219-4b13-846a-00c1568701aa

Field	Old	New
duplicateCount	1	0

52929d9-3733-460d-a285-43b0236fde4

Field	Old	New
duplicateCount	1	0


3146d7c5-7d4a-4816-9b02-d77e94452200

Field	Old	New
duplicateCount	1	0

2c302216-7212-41ca-835e-eda395364656

Field	Old	New
duplicateCount	2	0

Device Group Health Diff



Added	Removed	Modified
1	0	0

Statistics Changes

Field	Old	New
DeviceHealth.Red	0	1
DeviceHealth.TotalDevices	2	3

Device Group Added Devices

Name	State
r0	red

Device Group Removed Devices

None

Device Group Modified Devices

None

Device Facts Diff

Added Device Facts

None

Removed Device Facts

None

Modified Device Facts

None

A sample of a report diff for a network group is shown below.

DIFF

Header Diff

Field	Old	New
Name	S1-r1-N1	S1-r1-dut_chk
Group	N1	dut_chk
Type	network-group	device-group

Alert Data Diff



Added	Removed	Modified
23	1	0

Statistics Changes

Field	Old	New
SeverityCounts	Critical: 0 Major: 1 Minor: 0 Normal: 0	Critical: 0 Major: 18 Minor: 1 Normal: 4
StatusCounts	Ack: 0 Closed: 0 Open: 1 Shelved: 0 Expired: 0	Ack: 0 Closed: 4 Open: 19 Shelved: 0 Expired: 0
TotalAlerts	1	23
TotalMajorOpen	1	18
TotalMinorOpen	0	1
TotalNormalClosed	0	4

Added Alerts

Severity	Status	CreateTime	DuplicateCount	Environment	Service	Resource	Event	Text
major	open	2019-11-20T10:53:20.974Z	2	Production:dut_chk	system.cpu	r0	► Event	No data is being received on Sensor
major	open	2019-11-20T06:51:54.556Z	101	Production:dut_chk	system.memory	r1	► Event	No data is being received on Sensor
major	open	2019-11-20T06:51:54.529Z	101	Production:dut_chk	system.cpu	r1	► Event	No data is being received on Sensor
minor	open	2019-11-20T11:53:20.686Z	0	Production:dut_chk	system.cpu	r0	► Event	Routing Engine0 15min CPU utilization(71) is in medium range
normal	closed	2019-11-20T11:51:20.682Z	0	Production:dut_chk	system.cpu	r0	► Event	Routing Engine0 5min CPU utilization(53) is normal
normal	closed	2019-11-20T11:49:21.690Z	0	Production:dut_chk	system.processes	r0	► Event	ppt_11_80000010 cpu utilization(0), is normal
normal	closed	2019-11-20T11:49:20.692Z	0	Production:dut_chk	system.cpu	r0	► Event	Routing Engine0 CPU utilization(1) is normal
normal	closed	2019-11-20T11:49:20.688Z	0	Production:dut_chk	system.cpu	r0	► Event	Routing Engine0 1min CPU utilization(31) is normal
major	open	2019-11-20T10:53:40.815Z	1	Production:dut_chk	system.processes	r0	► Event	No data is being received on Sensor
major	open	2019-11-20T10:53:40.781Z	1	Production:dut_chk	system.storage	r0	► Event	No data is being received on Sensor
major	open	2019-11-20T10:53:35.781Z	1	Production:dut_chk	system.processes	r0	► Event	No data is being received on Sensor
major	open	2019-11-20T10:53:25.779Z	0	Production:dut_chk	system.storage	r2	► Event	No data is being received on Sensor
major	open	2019-11-20T10:53:21.006Z	0	Production:dut_chk	system.cpu	r2	► Event	No data is being received on Sensor
major	open	2019-11-20T10:53:20.943Z	0	Production:dut_chk	system.processes	r2	► Event	No data is being received on Sensor
major	open	2019-11-20T10:53:20.909Z	0	Production:dut_chk	system.cpu	r1	► Event	No data is being received on Sensor
major	open	2019-11-20T10:53:20.875Z	0	Production:dut_chk	system.storage	r1	► Event	No data is being received on Sensor
major	open	2019-11-20T10:53:20.846Z	0	Production:dut_chk	system.processes	r1	► Event	No data is being received on Sensor
major	open	2019-11-20T10:53:20.812Z	0	Production:dut_chk	system.processes	r2	► Event	No data is being received on Sensor
major	open	2019-11-20T10:53:20.780Z	0	Production:dut_chk	system.processes	r1	► Event	No data is being received on Sensor
major	open	2019-11-20T06:35:55.872Z	83	Production:dut_chk	system.cpu	r0	► Event	No data is being received on Sensor
major	open	2019-11-20T06:35:55.912Z	83	Production:dut_chk	system.memory	r0	► Event	No data is being received on Sensor
major	open	2019-11-20T06:33:57.147Z	1	Production:dut_chk	system.cpu	r0	► Event	Routing Engine1 5min CPU utilization(\$cpu-5min) exceed high threshold(80)
major	open	2019-11-20T06:33:57.143Z	1	Production:dut_chk	system.cpu	r0	► Event	Routing Engine1 15min CPU utilization(\$cpu-15min) exceed high threshold (75)

Removed Alerts

Severity	Status	CreateTime	DuplicateCount	Environment	Service	Resource	Event	Text
major	open	2019-11-20T06:33:10.688Z	1	Production:N1	ospf	-	► Event	Peer MTUs are not equal r0-mtu: \$r0-mtu -r1-mtu: \$r1-mtu

Modified Alerts

None

Network Group Health Diff

Field	Old	New
NetworkHealth.Name	N1	
NetworkHealth.State	red	

Device Facts Diff

Added Device Facts

None

Removed Device Facts

None

Modified Device Facts

None

Graph Diff

None

Release History Table

Release	Description
3.1.0	Capture a snapshot of a specific set of fields at a given point in time (HealthBot 3.1.0 and later)
2.1.0	Generate reports on demand (HealthBot 2.1.0 and later)
2.1.0	Compare (diff) two reports (HealthBot 2.1.0 and later)

Configure a Secure Data Connection for HealthBot Devices

HealthBot supports the following authentication methods to provide a secure data connection for HealthBot devices:

Authentication Method	Sensor Type	Description	Required HealthBot Security Parameters
Mutual SSL	OpenConfig	Client authenticates itself with the server and the server authenticates itself with the client.	<ul style="list-style-type: none"> Local certificates (includes the client certificate and client key) CA certificate Server common name
Server-side SSL	OpenConfig	Server authenticates itself with the client.	<ul style="list-style-type: none"> CA certificate Server common name
Public key SSH	iAgent	Authenticates users with password-protected SSH key files.	<ul style="list-style-type: none"> SSH key file Passphrase Username
Password	All	Authenticates users with a password.	<ul style="list-style-type: none"> Username Password

You can associate SSL or SSH certificates and keys with HealthBot devices through user-defined security profiles:

- [Configure Security Profiles for SSL and SSH Authentication on page 217](#)
- [Configure Security Authentication for a Specific Device or Device Group on page 218](#)

Configure Security Profiles for SSL and SSH Authentication

To configure security profiles for SSL and SSH authentication:

1. Click the **Settings > Security** option in the left-nav bar.
2. Click the add profile button for one of the following profiles and enter the required information:

Security Profile	Description of Parameters
CA	<p>Name—Enter profile name.</p> <p>Upload Certificate—Choose the CA certificate file and then click Open. The supported file extension is CRT.</p>
Local Certificates	<p>Name—Enter profile name.</p> <p>Upload Certificate—Choose the client certificate file and then click Open. The supported file extension is CRT.</p> <p>Upload Key—Choose the client key file and then click Open. The supported file extension is KEY.</p>
SSH Keys	<p>Name—Enter profile name.</p> <p>Upload Key File—Choose the private key file generated by ssh-keygen and then click Open.</p> <p>Passphrase—Enter the authentication passphrase.</p>

3. Click **Save** to save the configuration or click **Save and Deploy** to save and deploy the configuration.
4. Repeat Steps 4 and 5, as needed.
5. Apply the security profiles to a specific device or device group. For more details, see [“Configure Security Authentication for a Specific Device or Device Group”](#) on page 218.

Configure Security Authentication for a Specific Device or Device Group

1. Click the **Dashboard** option in the left-nav bar.
2. Click the name of the device or device group for which you want to configure security authentication. The device or device group profile pane appears, respectively.
3. Under **Authentication**, enter the required parameters for each applicable authentication method: Password, SSL, or SSH. All methods can be configured together on a single device or device group profile.

Authentication Method	Description of Parameters
Password	<p>Username—Enter the authentication username.</p> <p>Password—Enter the authentication password.</p>
SSL	<p>Server Common Name—Enter the 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—Enter the authentication username.</p>

*To edit or view details about saved security profiles, go to the Settings > Security page in the left-nav bar.

The following guidelines apply to the **Authentication** configuration:

- HealthBot decides which authentication method to apply to a device or device group based on which of the required security parameters are configured.
 - When more than one method is valid, HealthBot prioritizes SSL and SSH authentication over password-based authentication.
 - HealthBot prioritizes device-level settings over device group-level settings.
4. Click **Save** to save the configuration or click **Save and Deploy** to save and deploy the configuration.

Configure Data Summarization

You can improve the performance and disk space utilization of the HealthBot time series database (TSDB) by configuring data summarization methods to summarize the raw data collected by HealthBot. The data can be summarized as a function of time or when a change occurs.

For time-based data summarization, the raw data points are grouped together into user-defined time spans, and each group of data points is summarized into one data point using aggregate functions.

[Table 6 on page 219](#) provides a list of the supported data summarization algorithms and a description of their output:

Table 6: Descriptions of the Data Summarization Algorithms

Algorithm	Description of output
Latest	Value of the last data point collected within the time span.
Count	Total number of data points collected within the time span.
Mean	Average value of the data points collected within the time span.
Min	Minimum value of the data points collected within the time span.
Max	Maximum value of the data points within the time span.
On-change	Value of the data point whenever the value is different from the previous data point (occurs independently from the user-defined time span).
Stddev	Standard deviation of the data points collected within the time span.
Sum	Sum of the data points collected within the time span.

If no summarization algorithm is associated with the data, the following algorithms are used by default:

Data type	Data summarization algorithm
Float, integer, unsigned	Mean
Boolean, string	On-change

You can use data summarization profiles to apply specific summarization algorithms to the raw data collected by HealthBot for a specific device group:

- [Creating a Data Summarization Profile on page 220](#)
- [Applying Data Summarization Profiles to a Device Group on page 221](#)

Creating a Data Summarization Profile

To create a data summarization profile that can be applied to a device group:

1. Click the **Settings > Data Summarization Profiles** link in the left-nav bar.
2. Click the **+ Summarization Profile** button.
3. In the **Name** field, enter the name of the profile.
4. Under **Type Aggregate**, click the **+ Add Type Aggregate** button.
5. Choose a data type name and associate it with a data summarization algorithm using the drop-down lists. Data types include: string, integer, boolean, and float. Note: The algorithm configured for a specific sensor path name overrides the algorithm configured for the corresponding data type.
6. Repeat Step 5 and Step 6 for other data types, as needed. You can associate more than one algorithm with the same data type.
7. Under **Path Aggregate**, click the **+ Add Path Aggregate** button.
8. Enter a sensor path name, and associate it with a data summarization algorithm using the drop-down list. Note: The algorithm configured for a specific sensor path name overrides the algorithm configured for the corresponding data type.

You can enter a path name for a sensor that is not supported by HealthBot. For sensors supported by HealthBot, the path name must be entered in the following format:

Sensor	Path Name Format	Example
Open Config	<i>sensor-path</i>	/components/component/name
Native GPB	<i>sensor-name:sensor-path</i>	jnpr_qmon_extqueue_monitor_element_info:percentage
iAgent	<i>yaml-table-name:sensor-path</i>	REutilizationTable:15_min_cpu_idle

Sensor	Path Name Format	Example
SNMP	<i>snmp-table-name:sensor-path</i>	.1.3.6.1.2.1.2.2:jnxLED1Index ospfNbrTable:ospfNbrIpAddr
Syslog	<i>pattern-set: sensor-path</i>	interface_link_down:operational-status
Flow (NetFlow)	<i>template-name:sensor-path</i>	hb-ipfix-ipv4-template:sourceIPv4Address

- Repeat Step 8 and Step 9 for other sensor paths, as needed. You can associate more than one algorithm with the same path name.
- Click **Save** to save the configuration or click **Save and Deploy** to save and deploy the configuration.
- Apply the data summarization profile to a specific device group. For more details, see [“Applying Data Summarization Profiles to a Device Group” on page 221.](#)

Applying Data Summarization Profiles to a Device Group

After creating a data summarization profile, you can apply the profile to a specific device group to start summarizing the raw sensor data:

- Click the **Configuration > Device Group** option in the left-nav bar.
- Click the name of the device group for which you want to apply the data summarization profile.
- Click the **Edit Device Group (Pencil)** icon.
The **Edit <device-group-name>** window appears.
- Click the **Summarization** header to expand that section.,
- Enter the **Time Span** in seconds (s), minutes (m), hours (h), days (d), weeks (w), or years (y)
- 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** page under the **Settings** menu option in the left-nav bar.

If you select two or more profiles, the following guidelines apply:

- If the same data type or sensor path name is configured in two or more profiles, the associated algorithms will be combined.
- The table that stores the summarization output includes columns of summarized data for each algorithm associated with each data field collected by HealthBot. The naming convention for each column is as follows:

Number of algorithms associated with a data field	Column name for the summarized output
1	<i>field-name</i> Example: 5_sec_cpu_idle
2	<i>field-name_first-algorithm-name, field-name_ second-algorithm-name</i> Example: 5_sec_cpu_idle_MIN, 5_sec_cpu_idle_MAX
3	<i>field-name_first-algorithm-name, field-name_ second-algorithm-name, field-name_ third-algorithm-name...</i>

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

Modify the UDA and UDF Engines

IN THIS SECTION

- [Overview | 222](#)
- [How it Works | 223](#)
- [Usage Notes | 224](#)
- [Configuration | 225](#)

Overview

When creating rules, HealthBot includes the ability to run user-defined actions (UDAs) as part of a trigger. UDAs are essentially Python scripts that can be configured to be triggered by a HealthBot rule. For example,


```
optional arguments:
  -h, --help            show this help message and exit
  -s SCRIPT, --script SCRIPT
                        Run script in UDF engine
  --rollback, -r        Rollback UDF engine to original state
  --simulate            Run script in simulated UDF engine and show output
  --service SERVICE    Modify specific service UDF
```

The commands have three main options:

- Simulate—test a script (and view its output) in the simulated UDA/UDF engine environment without affecting the running HealthBot system
- Modify—modify the actual UDA/UDF engine using a script
- Rollback—revert to the original version of the UDA/UDF engine

Usage Notes

- The bash script will run in a container running Ubuntu OS Release 16.04 or 18.04; write the script accordingly.
- The script must be non-interactive; any questions must be pre-answered. For example, use the ‘-y’ option when installing a package using apt-get.
- If you prefer to copy the source packages of the dependency modules onto the HealthBot server so the engine can manually install them instead of downloading them from the Internet, place the required source packages in the `/var/local/healthbot/input` directory. Then within your bash script, point to the `/input` directory. For example, to use a file placed in `/var/local/healthbot/input/myfile.txt`, set the bash script to access it at `/input/myfile.txt`.
- Modifying the UDA/UDF engine more than once is *not* an incremental procedure; use a new bash script that includes both the original and new instructions, and re-run the modify procedure using the new script.
- UDA/UDF modifications are persistent across upgrades.

Configuration

As a best practice, we recommend that you use the following workflow:



This best-practice approach ensures that you first validate your script in the simulated environment before modifying the real engine.

NOTE: The examples below use the UDA engine; these procedures apply equally to the UDF engine.

NOTE: The procedure below assumes your HealthBot server is installed, including running the `sudo healthbot setup` command.

SIMULATE

Use the simulate feature to test your bash script in the simulated environment, without affecting the running HealthBot system.

To simulate modifying the UDA engine:

1. Enter the command `healthbot modify-uda-engine -s /<path>/<script-file> --simulate`.
2. The script runs and the output shows on screen, just as if you entered the script commands yourself.

```

user@HB-server:~$ healthbot modify-uda-engine -s /var/tmp/test-script.sh --simulate
Running /var/tmp/test-script.sh in simulated alerta engine..
Get:1 http://security.ubuntu.com/ubuntu xenial-security InRelease [109 kB]
...
Fetched 4296 kB in 15s (278 kB/s)
Reading package lists...
  
```

```
Building dependency tree...
Reading state information...
...
```

MODIFY

When you are satisfied with the simulation results, go ahead with the actual modification procedure.

To modify the UDA engine:

1. Load the desired bash script onto the HealthBot server.
2. If your HealthBot server is fully up and running, issue the command **healthbot stop** to stop the running services.
3. Run the command **healthbot modify-uda-engine -s /<path>/<script-file>**.

```
user@HB-server:~$ healthbot modify-uda-engine -s /var/tmp/test-script.sh
Running /var/tmp/test-script.sh in simulated alerta engine..
Success! See /tmp/.alerta_modification.log for logs
Please restart alerta by issuing 'healthbot start --device-group healthbot -s
alerta'
```

4. (Optional) As noted in the output, you can check the log file to further verify the script was loaded successfully.
5. Restart the alerta service using the command **healthbot start -s alerta**.
6. Once complete, verify that the alerta service is up and running using the command **healthbot status**.
7. To verify that the UDA engine has been updated, use the command **healthbot version -s alerta** and check that the healthbot_alerta container is using the **<version>-custom** tag.

```
user@HB-server:~$ healthbot version -s alerta
{'alerta': 'healthbot_alerta:2.1.0-custom'}
```

The UDA engine is now running with the installed dependencies as per the bash script.

ROLLBACK

If you have a need or desire to remove the changes to the engine, you can revert the engine to its original state.

To rollback the UDA engine:

1. Enter the command **healthbot modify-uda-engine --rollback**.

```
user@HB-server:~$ healthbot modify-uda-engine --rollback
Rolling back alerta engine to original state..
Successfully rolled back alerta engine
Please restart alerta by issuing 'healthbot start --device-group healthbot -s
alerta'
```

Note that it is not necessary to restart the alerta service at this point.

2. Once complete, verify that the alerta service is up and running using the command **healthbot status**.
3. To verify that the UDA engine has reverted back, use the command **healthbot version -s alerta** and check that the healthbot_alerta container is using the **<version>** tag.

```
user@HB-server:~$ healthbot version -s alerta
{'alerta': 'healthbot_alerta:2.1.0'}
```

The UDA engine is now running in its original state, with no additional installed dependencies.

Release History Table

Release	Description
3.2.0	Starting with HealthBot Release 3.2.0, the processing of UDF/UDA fields has been moved to a UDF farm.

Logs for HealthBot Services

HealthBot runs various services (such as iAgent, jtimon, and Telegraf) to monitor the health of the network and individual devices. Each of these HealthBot services runs independently in a containerized environment and produces its own set of log messages that are categorized by severity level. You can configure different levels of logs to collect and download.

Table 7 on page 228 lists the severity levels of the HealthBot services logs. The severity levels are listed in order from the highest severity (greatest effect on functionality) to the lowest. If you select a lower severity level, the logs for each of the higher severity levels will also be collected. The log level for all services is set to **error** by default.

Table 7: HealthBot Service Log Message Severity Levels

Severity Level	Description
Error	(Highest level) Conditions that require correction.
Warn	Conditions that warrant monitoring.
Info	Non-error conditions of interest.
Debug	(Lowest level) Debug messages.

This topic includes:

- [Configure Service Log Levels for a Device Group or Network Group on page 228](#)
- [Download Logs for HealthBot Services on page 229](#)

Configure Service Log Levels for a Device Group or Network Group

You can collect different severity levels of logs for the running HealthBot services of a device group or network group. To configure which log levels to collect:

1. Click the **Configuration > Device Group** option in the left-nav bar.
2. For a device group, click on the device group name from the list of DEVICE GROUPS.
For a network group, click on the network group name from the list of NETWORK GROUPS.
3. For a Device Group, click the **Edit Device Group (Pencil)** icon
For a Network Group, click the **Edit Network Group (Pencil)** icon
4. In the edit window that pops up, click the caret next to the **Logging Configuration** heading to display the configuration fields.

- From the drop-down list for **Global Log Level**, select the level of the log messages that you want to collect for every running HealthBot service for the device or network group. See [Table 7 on page 228](#) for a definition of the log severity levels. The level is set to **error** by default.
- In the **Log Level for specific services** section, 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.
- Click **Save** to save the configuration or click **Save and Deploy** to save and deploy the configuration.

Download Logs for HealthBot Services

You can choose to download the collected logs for:

- Every running HealthBot service for a specific device group or network group.
- A specific running HealthBot service for a specific device group or network group.
- Common HealthBot services that are running by default for the HealthBot application.

To download the logs for HealthBot services:

- Select the **Administration > Log Collection** option in the left-nav bar.
- Select the **Group Type**, **Group Name**, and **Service Name** of the logs that you want to download:

Parameter	Description
Group Type	Options are: device —Services running for a device group network —Services running for a network group common-services —Services running by default for the HealthBot application.
Group Name	<ul style="list-style-type: none"> • For device Group Type, select a device group name. • For network Group Type, select a network group name. • For common-services Group Type, select HealthBot.
Service Name	(Optional) Select the specific HealthBot service for which you want to download the logs. If no service is selected, the logs for every service listed will be downloaded.

3. Click **Download** to download the logs to a file on your server. The filename is **healthbot_logs.zip** by default.

Troubleshooting

IN THIS SECTION

- [HealthBot Self Test | 230](#)
- [Device Reachability Test | 232](#)
- [Ingest Connectivity Test | 234](#)
- [Debug No-Data | 236](#)

Starting with Release 2.1.0, HealthBot supports four verification and troubleshooting features:

- [“HealthBot Self Test” on page 230](#): Verifies that HealthBot is working properly
- [“Device Reachability Test” on page 232](#): Verifies that network devices are up and reachable via ping & SSH
- [“Ingest Connectivity Test” on page 234](#): Validates which ingest types are supported for a given device
- [“Debug No-Data” on page 236](#): Provides debugging when a device shows “No-data” in HealthBot GUI

You can access these features by navigating to **Administration** > **Debug** from the left-nav panel.

HealthBot Self Test

Overview

When setting up basic functionality in HealthBot, it can be challenging to diagnose problems. From installation, to device configuration, to adding devices and applying playbooks, when an issue occurs there are many possible areas to investigate.

Starting with HealthBot 2.1.0, the self-test tool validates the core functionality of HealthBot. To perform the self test, the tool performs a typical set of tasks:

- Adds a simulated device to HealthBot
- Creates a device group, and adds the device
- Creates a rule
- Creates and deploys a playbook
- Streams data from the simulated device
- Displays ongoing status in the dashboard

The self-test instance essentially acts as a fully working setup, running entirely within the HealthBot system. When testing is complete, the tool provides a report.

Other Uses for the Self Test Tool

In addition to validating the HealthBot installation, the self-test feature also provides:

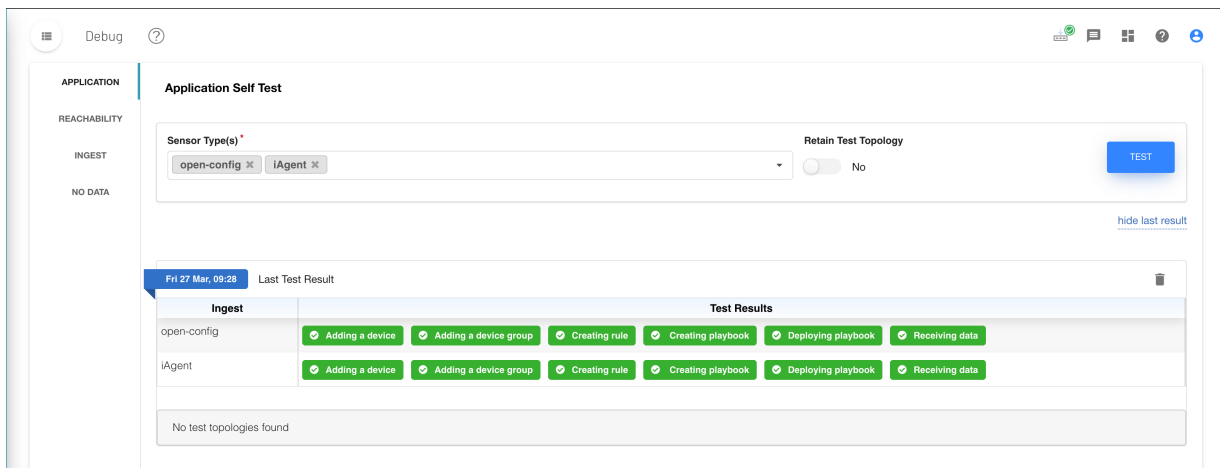
- An easy way to do a quick demo - the self test instance provides a simulated device connected to HealthBot, so you can demo HealthBot with no need to add a real device or apply playbooks.
- A good way for new users to get started - the self test auto-configures a simulated device connected to HealthBot, thereby eliminating the complexity of adding devices, applying playbooks, and so on.
- A 'running reference' - if there is an issue with real devices, you can use a self-test instance to help determine where the issue is; if the self-test instance is OK then the problem is not with the HealthBot system.

Usage Notes

- Currently this feature supports simulating devices to stream data for OpenConfig telemetry and iAgent (NETCONF).
- You can retain the self-test instance to act as a 'running reference', as noted above.
- The color coding for the test results is as follows:
 - Green = pass
 - Yellow = error (unable to test)
 - Red = fail
 - Any items with yellow or red status will include a message with more detail about the issue.
- Do not use the self-test tool when there are undeployed changes, as the self-test tool issues its own deploy during execution.
- Do not use rules, playbooks, devices, device-groups or other elements created by the self-test tool with real network devices.

How to Use the Self Test Tool

1. Navigate to the **Administration > Debug** page from the left-nav panel, and select the **APPLICATION** tab.
2. Select the desired sensor type(s) from drop-down menu.
3. Click the **Test** button.
4. After a few moments, the test results appear.



The example above shows that both sensors are working as expected.

Device Reachability Test

Overview

In early versions of HealthBot, there was no way to easily determine whether the devices you added were up and reachable; you would need to get through the entire setup procedure - add the device, setup a device group, apply playbooks, monitor devices - at which point the health pages would indicate “no data” indicating that the setup did not work correctly. Furthermore, “no data” does not indicate whether the problem is a reachability issue or data streaming issue.

Starting with HealthBot 2.1.0, the device reachability tool can verify connectivity to a device. The tool performs tests using ping and SSH. HealthBot uses the device’s IP address or host name, based on what was configured when adding the device.

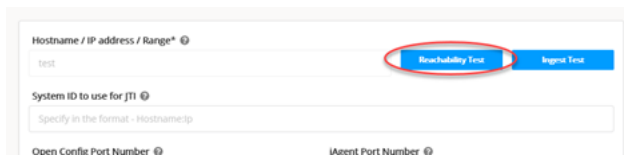
Usage Notes

- While this feature is generally intended to help troubleshoot device onboarding, you can use it any time to check device reachability.
- The color coding for the test results is as follows:
 - Green = pass
 - Yellow = error (unable to test)
 - Red = fail
 - Any items with yellow or red status will include a message with more detail about the issue.

How to Use the Device Reachability Tool

1. To access the tool:

- Navigate to the **Administration > Debug** page from the left-nav panel, and select the **REACHABILITY** tab.



The screenshot shows a web interface for the Device Reachability Tool. It features several input fields: 'Hostname / IP address / Range*' with a 'test' button, 'System ID to use for JT1' with a placeholder 'Specify in the format - Hostname:ip', 'Open Config Port Number', and 'Agent Port Number'. A blue button labeled 'Reachability Test' is highlighted with a red circle, and another blue button labeled 'Logout Test' is visible to its right.

- Or, on the Dashboard page click the desired device in the device list widget, and in the pop-up window click the **REACHABILITY TEST** button.

NOTE: The **REACHABILITY TEST** button does not appear when first adding the device.

2. In the Device Reachability tool, select the desired device from drop-down menu.

3. Click the **Test** button.
4. After a few moments, the test results appear.

Device Reachability

Device Group

All
▼

Device *

ix-vmx-02
▼

TEST

[hide last result](#)

Thu 26 Mar, 21:04

Last Test Result

Device Name	Ping	SSH
ix-vmx-02	✔ PASS	✔ PASS

CLOSE

The example above shows that the ping test was successful, but the SSH test failed.

Ingest Connectivity Test

Overview

In early versions of HealthBot, there was no way to easily determine which ingest methods were supported for a given device; you also had no way to know whether the configured ingest method successfully established a connection with the network device.

Starting with HealthBot 2.1.0, the ingest connectivity tool can verify ingest methods where HealthBot initiates the connection, such as OpenConfig, iAgent, and SNMP. HealthBot does not test UDP-based ingest methods, such as syslog and Native GPB, as the UDP parameters are common to a device group and not specific to a device.

HealthBot validates each supported ingest method in its own way:

- OpenConfig: Establishes a gRPC connection with the device using its IP/host name, gRPC port, and credentials
- iAgent: Establishes a NETCONF session with the device using its IP/host name, NETCONF port, and credentials

- **SNMP:** Executes a simple SNMP GET command; expects to get a reply from the device

This tool provides multiple benefits:

- It helps to identify when there might be missing configuration on the network device side.
- It helps you choose appropriate playbooks and rules that use sensors compatible with the supported ingest methods.
- It helps to identify ingest connectivity issues early on, rather than troubleshoot the “no-data” issue described in the previous section.


Usage Notes

- While this feature is generally intended to help troubleshoot device onboarding, you can use it any time to check ingest connectivity.
- The color coding for the test results is as follows:
 - Green = pass
 - Yellow = error (unable to test)
 - Red = fail
 - Any items with yellow or red status will include a message with more detail about the issue.


How to Use the Ingest Connectivity Tool

1. To access the tool:
 - Navigate to the **Administration > Debug** page from the left-nav panel, and click the **INGEST** tab.
 - Or, on the Dashboard page click the desired device in the device list widget, and in the pop-up window click the **INGEST TEST** button.



Edit test

Hostname / IP address / Range* 

test Troubability Test **Ingest Test**

System ID to use for JTI 

Specify in the format - Hostname:ip

Open Config Port Number  Agent Port Number 

NOTE: The **Ingest Test** button does not appear when first adding the device.

2. In the Ingest Connectivity tool, select the desired device from drop-down menu

3. Click the **Test** button.
4. After a few moments, the test results appear.

Ingest Connectivity

Sensor Type(s) *

open-config ✕ iAgent ✕ snmp ✕

Device Group **Device ***

All ix-vmx-02 TEST

[hide last result](#)

Thu 26 Mar, 21:05

 Last Test Result

Device	Sensors
ix-vmx-02	<div style="display: flex; gap: 5px;"> <div style="background-color: #28a745; color: white; padding: 2px 5px; border-radius: 3px; display: flex; align-items: center; gap: 2px;"> ✓ open-config </div> <div style="background-color: #28a745; color: white; padding: 2px 5px; border-radius: 3px; display: flex; align-items: center; gap: 2px;"> ✓ iAgent </div> <div style="background-color: #28a745; color: white; padding: 2px 5px; border-radius: 3px; display: flex; align-items: center; gap: 2px;"> ✓ snmp </div> </div>

CLOSE

The example above shows that iAgent is supported, OpenConfig is not supported, and SNMP encountered an error.

Debug No-Data

Overview

One of the most common problems that a HealthBot users face is “How to debug no-data?”. Determining the root cause is challenging as the issue can occur for a variety of reasons, including:

- Device not reachable
- Device not sending data
- Firewall blocking connections
- Ingest connectivity settings mismatch
- Rule frequency/trigger interval is configured less than the router response time
- HealthBot services not running

In early versions of HealthBot, you needed to manually look for the problem, checking add-device parameters, reviewing logs, checking the device configuration, and so on.

Starting with HealthBot 2.1.0, the debug no-data tool helps to determine why a device or rule is showing a status of “no-data”. The tool takes a sequential, step-by-step approach to determine at which stage incoming data is getting dropped or blocked, as follows:

- HealthBot Services
 - Verify that all common and device group-related services are up and running
- Device Reachability
 - Test connectivity to device using ping and SSH
- Ingest Connectivity
 - Verify that the configured ingest session is established
- Raw Data Streaming
 - Verify whether the ingest is receiving any raw data from the devices
 - Likely to be OK if device connectivity is OK
- Field Processing
 - Within rules, verify that the fields working properly, and that the field information is populated in the database
- Trigger Processing
 - Within rules, verify that the trigger settings working as intended, and status information is populated in the database
- API Verification
 - Check for API timeouts that might be affecting the GUI

Usage Notes

- The tool runs through the entire sequence of checks, regardless of any issues along the way.
- The test results provide root cause information and advise where to focus your troubleshooting efforts.
- While this feature is generally intended to debug a device when it is marked as no-data, you can use it any time to verify that deployed rules are receiving data.
- This tool does not support rules using a syslog sensor, as the sensor data is event driven and not periodic.
- The color coding for the test results is as follows:

- Green = pass
- Yellow = error (unable to test)
- Red = fail
- Any items with yellow or red status will include a message with more detail about the issue.

How to Use the Debug No-Data Tool

1. To access the tool:

- On any device health page, click a “no data” tile.



- Or, navigate to the **Administration > Debug** page from the left-nav panel, and select the NO DATA tab.
2. In the Debug No-Data tool, select the desired device group, device, and one or more rules from the drop-down menu

- 3. Click the **Debug** button.
- 4. After a few moments, the test results appear.

Debug No-Data

Device Group * Device * Topic *

Rule(s) **TEST**

Only topics and rules of running playbook instances are displayed here

[hide last result](#)

Thu 26 Mar, 22:07 Last Test Result

Device Group ix-lab-group Device ix-vmx-02 Topic interface.statistics

▶ HealthBot Services	✓
▶ Device Reachability	✓
▶ Ingest Connectivity	✓
▶ Data Streaming	✓
▶ Field Processing	✓
▶ Trigger Processing	✗
▶ API Verification	✓

CLOSE

The example above shows that one common services is not working properly.

HealthBot Configuration – Backup and Restore

IN THIS SECTION

- [Back Up the Configuration | 240](#)
- [Restore the Configuration | 240](#)
- [Backup or Restore the Time Series Database \(TSDB\) | 241](#)

Back Up the Configuration

To back up the current HealthBot configuration:

1. Select the **Administration > Backup** option from the left-nav bar.
2. Check the check box for the configuration that you want to backup.

If you also want to back up any configuration changes that have not yet been deployed, toggle the **Backup Undeployed Configuration** switch to the right.

3. Click **Backup**.

Back Up Helper Files

Helper files are the python, yaml, or other externally created files whose tables are referenced in iAgent sensor definitions within rule definitions.

1. Select the **Administration > Backup** option from the left-nav bar.
2. Click the **BACKUP HELPER FILE** button.

HealthBot creates a tar archive that you can save to your computer.

Restore the Configuration

To restore the HealthBot configuration to a previously backed up configuration:

1. Select the **Administration > Restore** option from the left-nav bar.
2. Click the **Choose File** button.
3. Navigate to the backup file from which you want to restore the configuration, and click **Open**.

Once opened, a list of backup configuration sections appears as buttons under **Select the configuration backup file** heading. By default, all configuration elements in the selected backup file are selected to restore.

4. (Optional) From the list of buttons, you can deselect (change the “-” to “+”) individual files from restoring by clicking on the button.
5. Click **RESTORE CONFIGURATION & DEPLOY** to restore and deploy the configuration.

Restore Helper File

To restore previously backed up helper file archives:

1. Select the **Administration > Restore** option from the left-nav bar.
2. Click the **Choose File** button directly above the second line, “Select the helper backup file”.
3. Locate the backed up helper file in the file browser and click **Open**.
4. Click the **RESTORE CONFIGURATION** button to restore the helper files to HealthBot, or the **RESTORE CONFIGURATION & DEPLOY** button to restore the helper files to their original location within HealthBot.

Backup or Restore the Time Series Database (TSDB)

Starting with HealthBot Release 3.2.0, you can backup and restore the TSDB separately from other configuration elements. The backup and restore operations for the TSDB are only available through the HealthBot CLI. The backup and restore commands are invoked by using a predefined python script, **healthbot.py**. You must have root access to the CLI interface of the HealthBot server in order to issue these commands.

An environment variable must be set on the HealthBot server prior to any backup or restore operation. The following shows how to set this variable.

```
export HB_EXTRA_MOUNT1=/root/.kube/config
```

NOTE: In the example above, *HB_EXTRA_MOUNT1* is a variable. As such, it could be given any name that you want.

The following shows the generic command and the optional arguments (in square brackets) available. Each option is then described and an example is provided.

```
healthbot tsdb (backup|restore) [-h] [--database DATABASE] [--all] --path PATH
```

The required arguments for the **healthbot tsdb** command are:

- **backup**—perform a backup operation
- **restore**—perform a restore operation
- **--path PATH**—create the backup file or restore the database(s) from the container at **PATH** where **PATH** is *HB_EXTRA_MOUNT1*.

Optional Arguments

```
-h, --help show this help message and exit
--database DATABASE Takes backup (or restore) of the given list of databases. Either
database or all flag must be configured
--all Takes backup (or restores) of all the databases. Either database or
all flag must be configured
```

Example – Backup the TSDB and Store it in *HB_EXTRA_MOUNT3*

```
healthbot tsdb backup --path HB_EXTRA_MOUNT3
```

Example – Restore the TSDB from *HB_EXTRA_MOUNT2*

```
healthbot tsdb restore --path HB_EXTRA_MOUNT2
```

Release History Table

Release	Description
3.2.0	Starting with HealthBot Release 3.2.0, you can backup and restore the TSDB separately from other configuration elements