Administrators Guide

OpenNMS Horizon 18.0.0, Last updated 2016-05-09 15:13:05 EDT

Table of Contents

Chapter 1. Data Choices

The `Data Choices` module collects and publishes anonymous usage statistics to https://stats.opennms.org.

When a user with the `Admin` role logs into the system for the first time, they will be prompted as to whether or not they want to opt-in to publish these statistics. Statistics will only be published once an `Administrator` has opted-in.

Usage statistics can later be disabled by accessing the 'Data Choices' link in the 'Admin' menu.

When enabled, the following anonymous statistics will be collected and publish on system startup and every 24 hours after:

- System ID (a randomly generated UUID)
- OpenNMS Horizon Release
- OpenNMS Horizon Version
- OS Architecture
- OS Name
- OS Version

1. Number of Alarms in the `alarms` table
2. Number of Events in the `events` table
3. Number of IP Interfaces in the `ipinterface` table
4. Number of Nodes in the `node` table
5. Number of Nodes, grouped by System OID

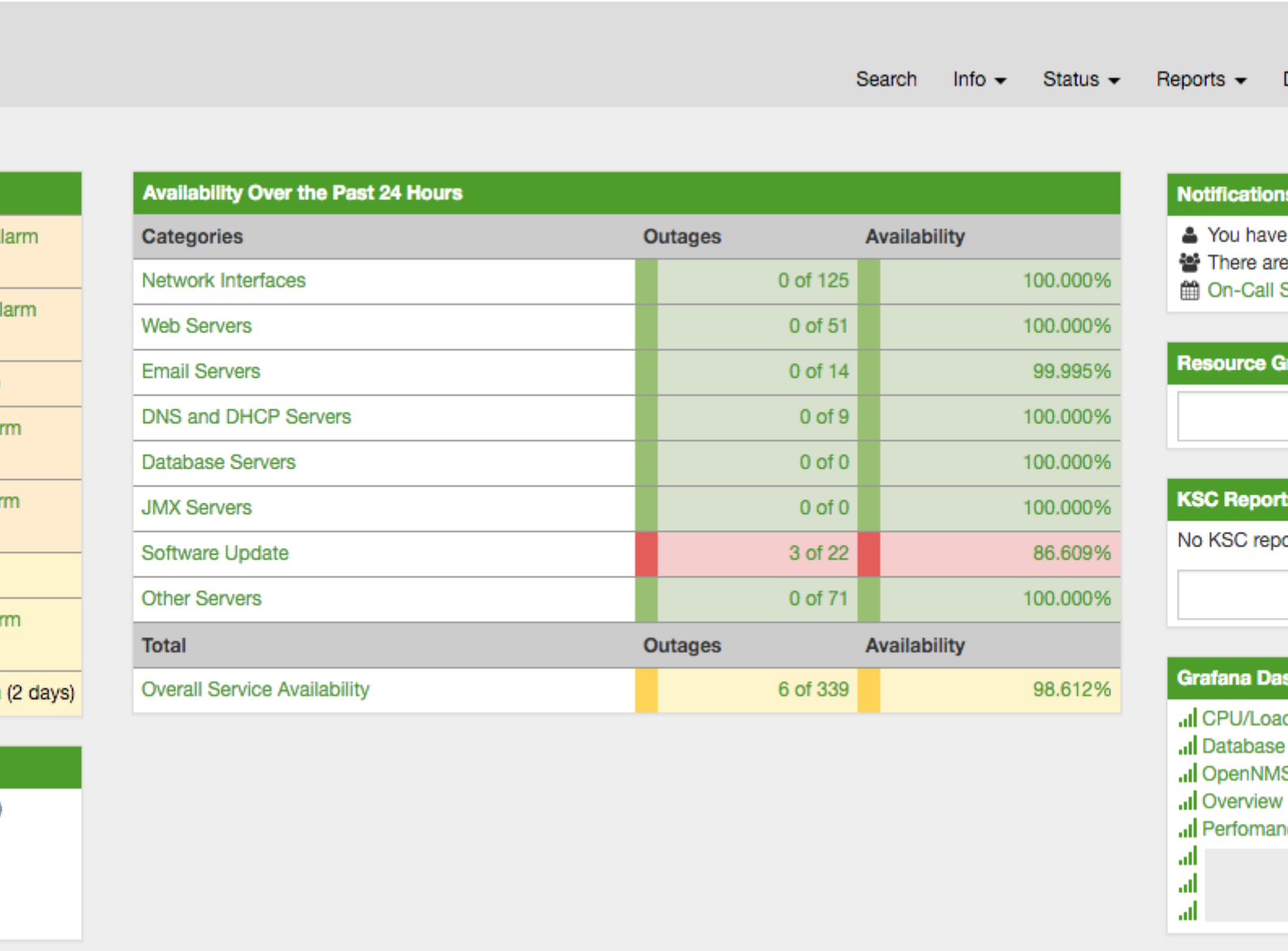
Chapter 2. Administrative Webinterface

2.1. Grafana Dashboard Box

Grafana provides an API key which gives access for 3rd party application like *OpenNMS Horizon*. The *Grafana Dashboard Box* on the start page shows dashboards related to *OpenNMS Horizon*. To filter relevant dashboards, you can use a *tag* for dashboards and make them accessible. If no *tag* is provided all dashboards from *Grafana* will be shown.

The feature is by default deactivated and is configured through `opennms.properties`. Please note that this feature works with the *Grafana API v2.5.0*.

Quick access to Grafana dashboards from the OpenNMS Horizon start page



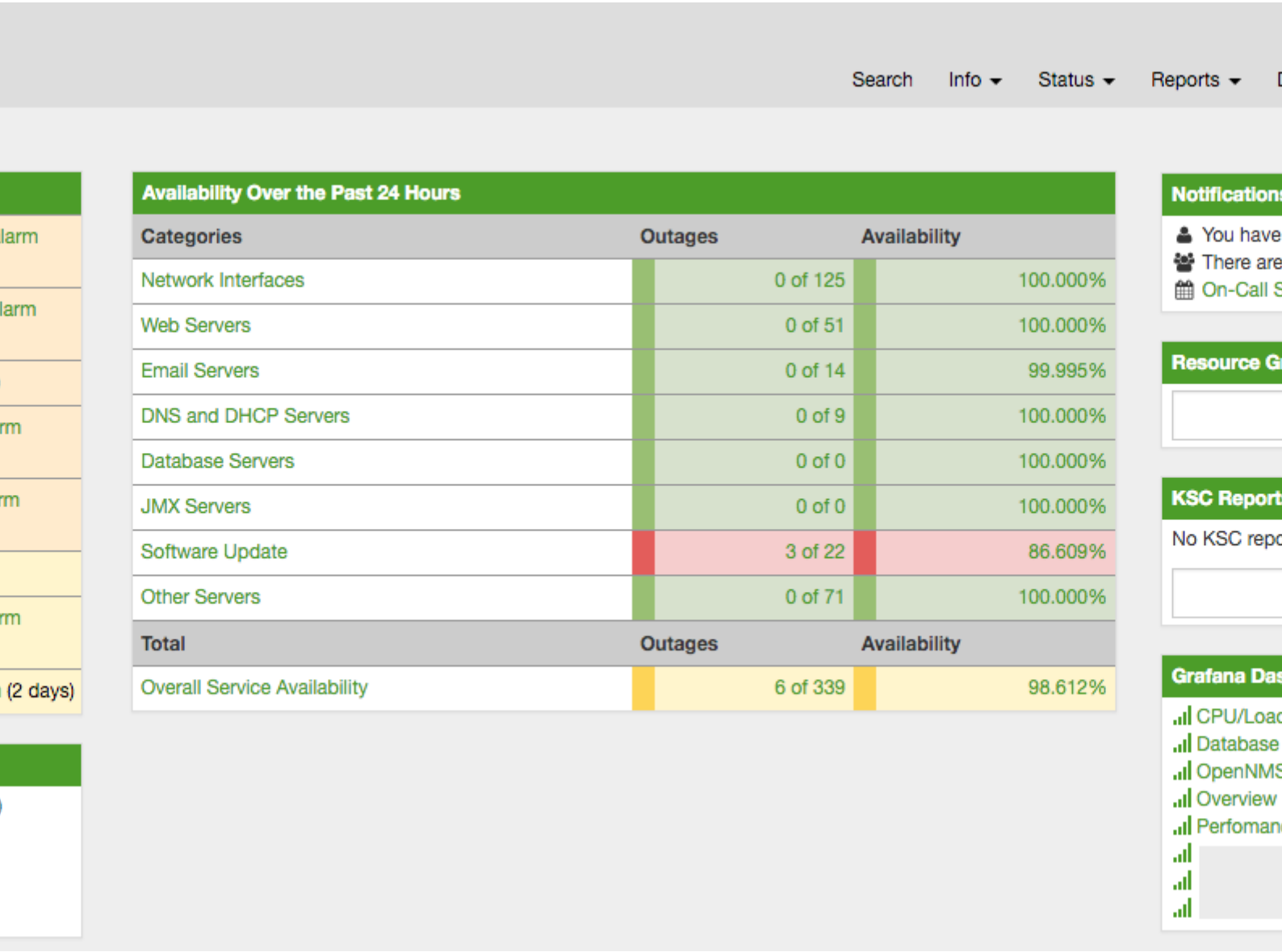


Table 1. Grafana Dashboard configuration properties

Name	Type	Description	Default
`org.opennms.grafanaBox.show`	*Boolean*	This setting controls whether a grafana box showing the available dashboards is placed on the landing page. The two valid options for this are `true` or `false`.	`false`
`org.opennms.grafanaBox.hostname`	*String*	If the box is enabled you also need to specify hostname of the *Grafana* server	`localhost`
`org.opennms.grafanaBox.port`	*Integer*	The port of the *Grafana* server ReST API	`3000`
`org.opennms.grafanaBox.apiKey`	*String*	The API key is needed for the ReST calls to work	

Name	Type	Description	Default
`org.opennms.grafanaBox.tag`	*String*	When a *tag* is specified only dashboards with this given *tag* will be displayed. When no *tag* is given all dashboards will be displayed	
`org.opennms.grafanaBox.protocol`	*String*	The protocol for the ReST call can also be specified	`http`
`org.opennms.grafanaBox.connectionTimeout`	*Integer*	Timeout in milliseconds for getting information from the *Grafana* server	`500`
`org.opennms.grafanaBox.soTimeout`	*Integer*		`500`

If you have *Grafana* behind a proxy it is important the `org.opennms.grafanaBox.hostname` is reachable. This host name is used to generate links to the *Grafana* dashboards.

The process to generate an *Grafana API Key* can be found in the HTTP API documentation. Copy the API Key to `opennms.properties` as `org.opennms.grafanaBox.apiKey`.

2.2. Operator Board

In a network operation center *(NOC)* the *Ops Board* can be used to visualize monitoring information. The monitoring information for various use-cases are arranged in configurable *Dashlets*. To address different user groups it is possible to create multiple *Ops Boards*.

There are two visualisation components to display *Dashlets*:

- *Ops Panel*: Shows multiple *Dashlets* on one screen, e.g. on a NOC operators workstation
- *Ops Board*: Shows one *Dashlet* at a time in rotation, e.g. for a screen wall in a NOC

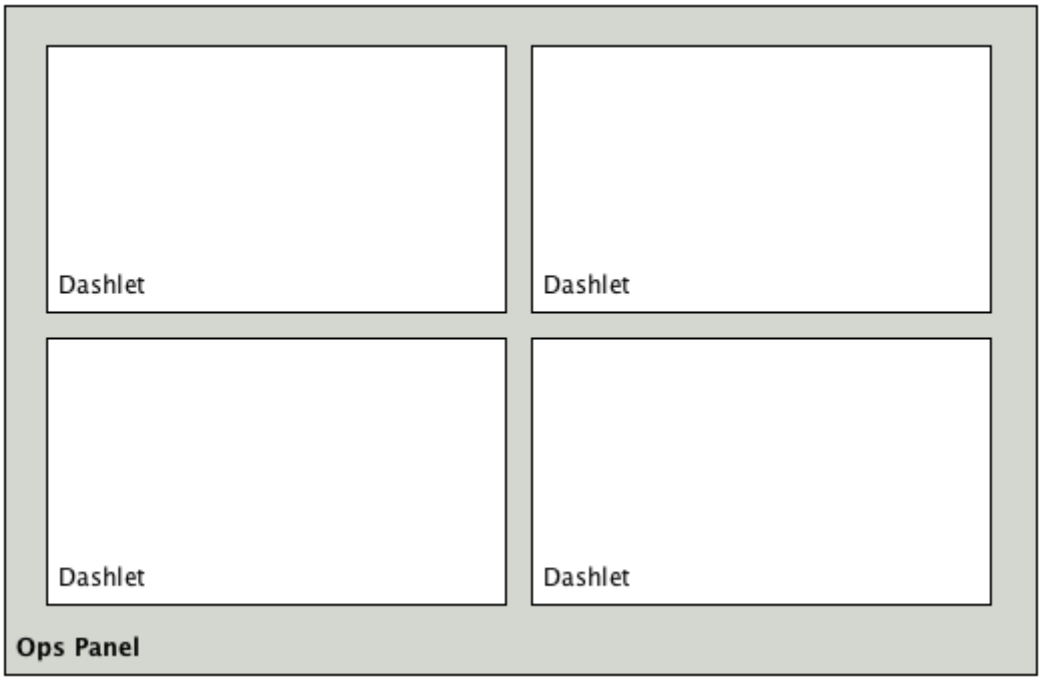


Figure 1. Concept of Dashlets displayed in Ops Panel

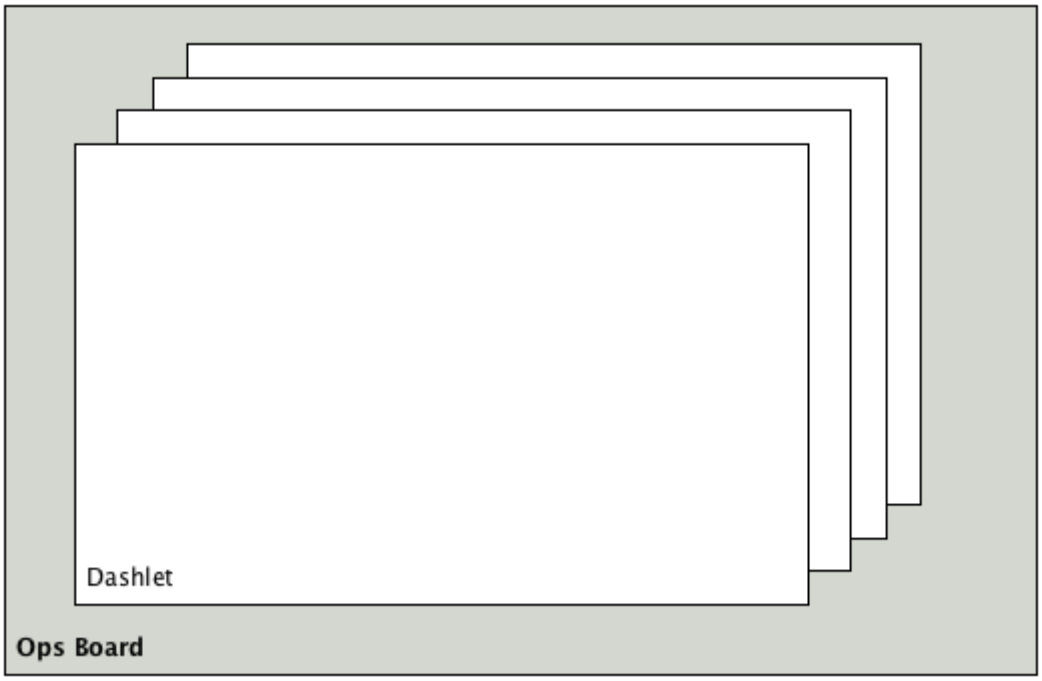


Figure 2. Concept to show Dashlets in rotation on the Ops Board

2.2.1. Configuration

To create and configure *Ops Boards* administration permissions are required. The configuration section is in admin area of OpenNMS Horizon and named *Ops Board Config Web Ui*.

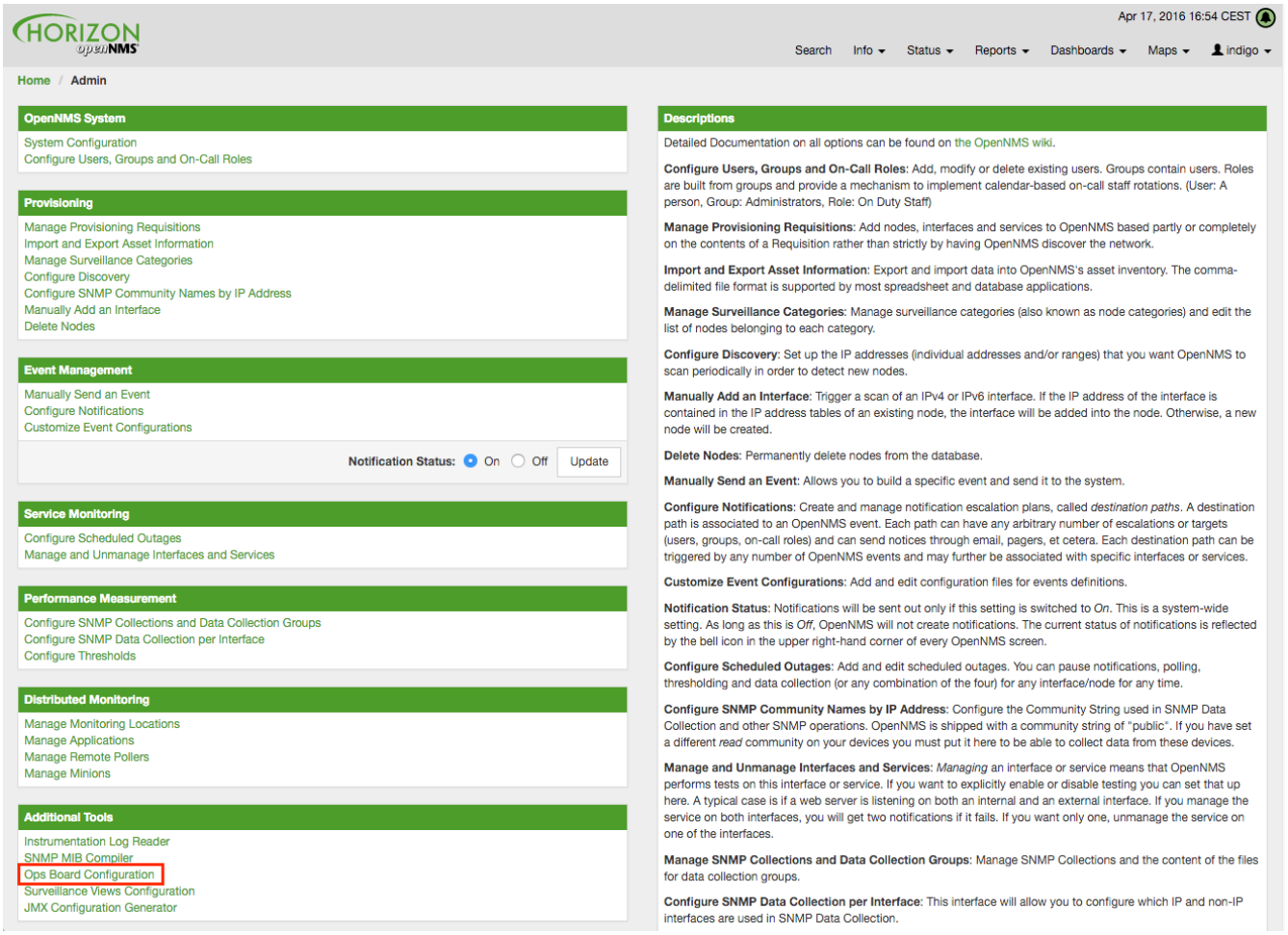


Figure 3. Navigation to the Ops Board configuration

Create or modify *Ops Boards* is described in the following screenshot.

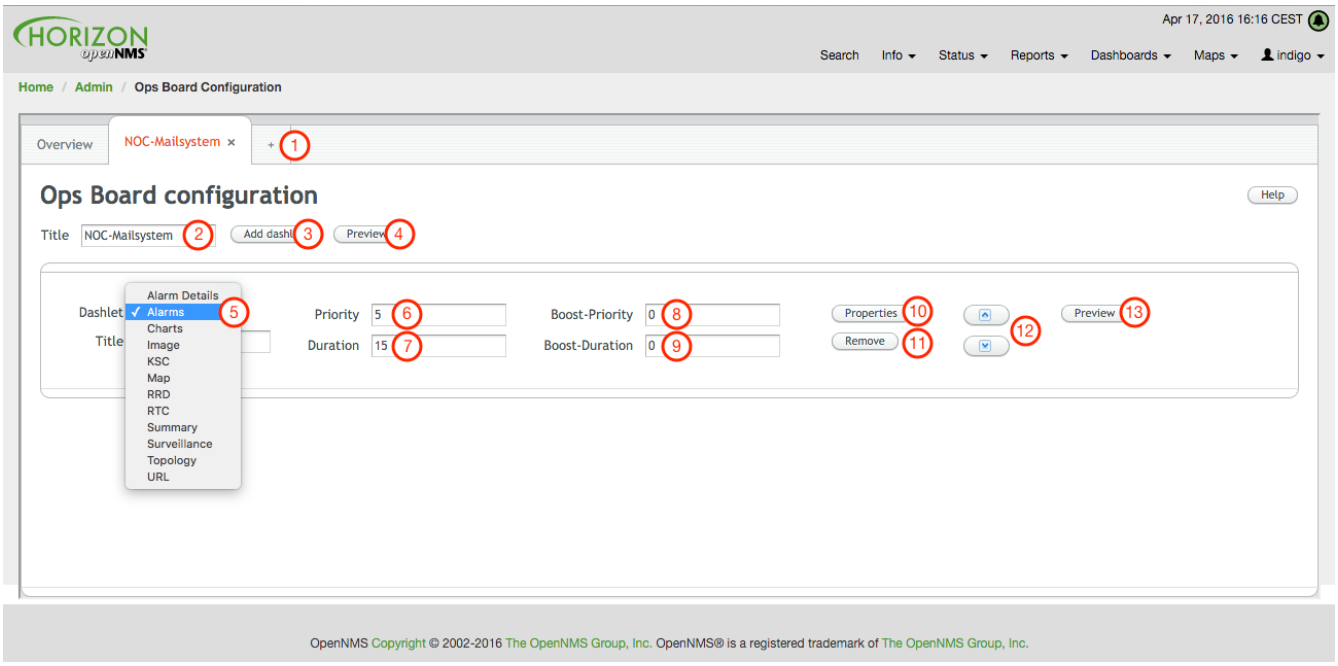


Figure 4. Adding a Dashlet to an existing Ops Board

1. Create a new *Ops Board* to organize and arrange different *Dashlets*
2. The name to identify the *Ops Board*
3. Add a *Dashlet* to show OpenNMS Horizon monitoring information

4. Show a preview of the whole *Ops Board*
5. List of available *Dashlets*
6. *Priority* for this *Dashlet* in *Ops Board* rotation, lower priority means it will be displayed more often
7. *Duration* in seconds for this *Dashlet* in the *Ops Board* rotation
8. Change *Priority* if the *Dashlet* is in alert state, this is optional and maybe not available in all *Dashlets*
9. Change *Duration* if the *Dashlet* is in alert state, it is optional and maybe not available in all *Dashlets*
10. Configuration properties for this *Dashlet*
11. Remove this *Dashlet* from the *Ops Board*
12. Order *Dashlets* for the rotation on the *Ops Board* and the tile view in the *Ops Panel*
13. Show a preview for the whole *Ops Board*

The configured *Ops Board* can be used by navigating in the main menu to *Dashboard → Ops Board*.

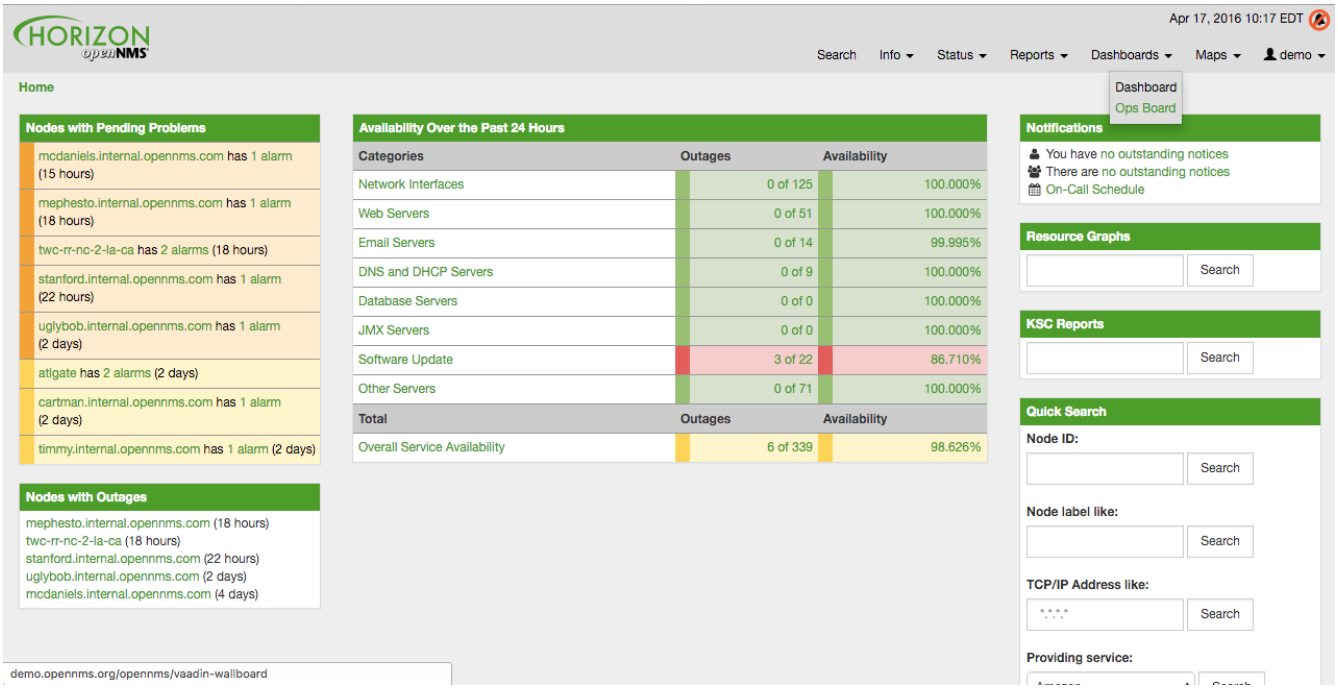


Figure 5. Navigation to use the Ops Board

2.2.2. Dashlets

Visualization of information is implemented in *Dashlets*. The different *Dashlets* are described in this section with all available configuration parameter.

To allow filter information the *Dashlet* can be configured with a generic Criteria Builder.

Alarm Details

This *Alarm-Details Dashlet* shows a table with alarms and some detailed information.

Table 2. Information of the alarms

Field	Description
Alarm ID	OpenNMS Horizon ID for the alarm
Severity	Alarm severity (Cleared, Indeterminate, Normal, Warning, Minor, Major, Critical)
Node label	Node label of the node where the alarm occurred
Alarm count	Alarm count based on reduction key for deduplication
Last Event Time	Last time the alarm occurred
Log Message	Reason and detailed log message of the alarm

The *Alarm Details Dashlet* can be configured with the following parameters.

Boost support	Boosted Severity
Configuration	Criteria Builder

Alarms

This *Alarms Dashlet* shows a table with a short alarm description.

Table 3. Information of the alarm

Field	Description
Time	Absolute time since the alarm appeared
Node label	Node label of the node where the alarm occurred
UEI	OpenNMS Horizon *Unique Event Identifier* for this alarm

The *Alarms Dashlet* can be configured with the following parameters.

Boost support	Boosted Severity
Configuration	Criteria Builder

Charts

This *Dashlet* displays an existing Chart.

Boost support	false
`Chart`	Name of the existing chart to display
`Maximize Width`	Rescale the image to fill display width
`Maximize Height`	Rescale the image to fill display height

Image

This *Dashlet* displays an image by a given URL.

Boost support	false
`imageUrl`	URL with the location of the image to show in this *Dashlet*
`maximizeHeight`	Rescale the image to fill display width
`maximizeWidth`	Rescale the image to fill display height

KSC

This *Dashlet* shows an existing KSC report. The view is exact the same as the *KSC report* is build regarding order, columns and time spans.

Boost support	false
`KSC-Report`	Name of the KSC report to show in this *Dashlet*

Map

This *Dashlet* displays the geographical map.

Boost support	false
`search`	Predefined search for a subset of nodes shown in the geographical map in this *Dashlet*

RRD

This *Dashlet* shows one or multiple RRD graphs. It is possible to arrange and order the RRD graphs in multiple columns and rows. All RRD graphs are normalized with a given width and height.

Boost support	false
`Columns`	Number of columns within the *Dashlet*
`Rows`	Number of rows with the *Dashlet*
`KSC Report`	Import RRD graphs from an existing KSC report and re-arrange them.
`Graph Width`	Generic width for all RRD graphs in this *Dashlet*
`Graph Height`	Generic height for all RRD graphs in this *Dashlet*
`Timeframe value`	Number of the given `Timeframe type`
`Timeframe type`	Minute, Hour, Day, Week, Month and Year for all RRD graphs

RTC

This *Dashlet* shows the configured SLA categories from the OpenNMS Horizon start page.

Boost support	false
-	-

Summary

This *Dashlet* shows a trend of incoming alarms in given time frame.

Boost support	Boosted Severity
`timeslot`	Time slot in seconds to evaluate the trend for alarms by severity and *UEI*.

Surveillance

This *Dashlet* shows a given Surveillance View.

Boost support	false
`viewName`	Name of the configured *Surveillance View*

Topology

This *Dashlet* shows a Topology Map. The *Topology Map* can be configured with the following parameter.

Boost support	false
`focusNodes`	Which node(s) is in focus for the topology
`provider`	Which topology should be displayed, e.g. Linkd, VMware
`szl`	Set the zoom level for the topology

URL

This *Dashlet* shows the content of a web page or other web application, e.g. other monitoring systems by a given URL.

Boost support	false
`password`	Optional password if a basic authentication is required
`url`	URL to the web application or web page
`username`	Optional username if a basic authentication is required

2.2.3. Boosting *Dashlet*

The behavior to boost a *Dashlet* describes the behavior of a *Dashlet* showing critical monitoring information. It can raise the priority in the *Ops Board* rotation to indicate a problem. This behavior can be configured with the configuration parameter *Boost Priority* and *Boost Duration*. These to configuration parameter effect the behavior on the *Ops Board* in rotation.

- *Boost Priority*: Absolute priority of the *Dashlet* with critical monitoring information.
- *Boost Duration*: Absolute duration in seconds of the *Dashlet* with critical monitoring information.

2.2.4. Criteria Builder

The *Criteria Builder* is a generic component to filter information of a *Dashlet*. Some *Dashlets* use this component to filter the shown information on a *Dashlet* for certain use case. It is possible to combine multiple *Criteria* to display just a subset of information in a given *Dashlet*.

Table 4. Generic Criteria Builder configuration possibilities

Restriction	Property	Value 1	Value 2	Description
`Asc`	-	-	-	ascending order
`Desc`	-	-	-	descending order
`Between`	database attribute	*String*	*String*	Subset of data between value 1 and value 2
`Contains`	database attribute	*String*	-	Select all data which contains a given text string in a given database attribute
`Distinct`	database attribute	-	-	Select a single instance
`Eq`	database attribute	*String*	-	Select data where attribute equals (`==`) a given text string
`Ge`	database attribute	*String*	-	Select data where attribute is greater equals than (`>=`) a given text value
`Gt`	database attribute	*String*	-	Select data where attribute is greater than (`>`) a given text value
`Ilike`	database attribute	*String*	-	*unknown*
`In`	database attribute	*String*	-	*unknown*
`Iplike`	database attribute	*String*	-	Select data where attribute matches an given IPLIKE expression
`IsNull`	database attribute	-	-	Select data where attribute is null
`IsNotNull`	database attribute	-	-	Select data where attribute is **not** null
`IsNotNull`	database attribute	-	-	Select data where attribute is **not** null
`Le`	database attribute	*String*	-	Select data where attribute is less equals than (`<=`) a given text value

Restriction	Property	Value 1	Value 2	Description
`Lt`	database attribute	*String*	-	Select data where attribute is less than (<) a given text value
`Le`	database attribute	*String*	-	Select data where attribute is less equals than (⇐) a given text value
`Like`	database attribute	*String*	-	Select data where attribute is like a given text value similar to SQL `like`
`Limit`	-	*Integer*	-	Limit the result set by a given number
`Ne`	database attribute	*String*	-	Select data where attribute is not equals (`!=`) a given text value
`Not`	database attribute	*String*	-	*unknown* difference between `Ne`
`OrderBy`	database attribute	-	-	Order the result set by a given attribute

2.3. JMX Configuration Generator

OpenNMS Horizon implements the *JMX* protocol to collect long term performance data for *Java* applications. There are a huge variety of metrics available and administrators have to select which information should be collected. The *JMX Configuration Generator Tools* is build to help generating valid complex *JMX* data collection configuration and *RRD graph* definitions for *OpenNMS Horizon*.

This tool is available as CLI and a web based version.

2.3.1. Web based utility

Complex *JMX* data collection configurations can be generated from a web based tool. It collects all available *MBean Attributes* or *Composite Data Attributes* from a *JMX* enabled Java application.

The workflow of the tool is:

1. Connect with *JMX* or *JMXMP* against a *MBean Server* provided of a *Java* application
2. Retrieve all *MBean* and *Composite Data* from the application
3. Select specific *MBeans* and *Composite Data* objects which should be collected by *OpenNMS Horizon*
4. Generate *JMX Collectd* configuration file and *RRD graph* definitions for *OpenNMS Horizon* as downloadable archive

The following connection settings are supported:

- Ability to connect to *MBean Server* with *RMI* based *JMX*
- Authentication credentials for *JMX* connection

- Optional: JMXMP connection

The web based configuration tool can be used in the *OpenNMS Horizon Web Application* in administration section *Admin → JMX Configuration Generator*.

Configure JMX Connection

At the beginning the connection to an *MBean Server* of a *Java* application has to be configured.

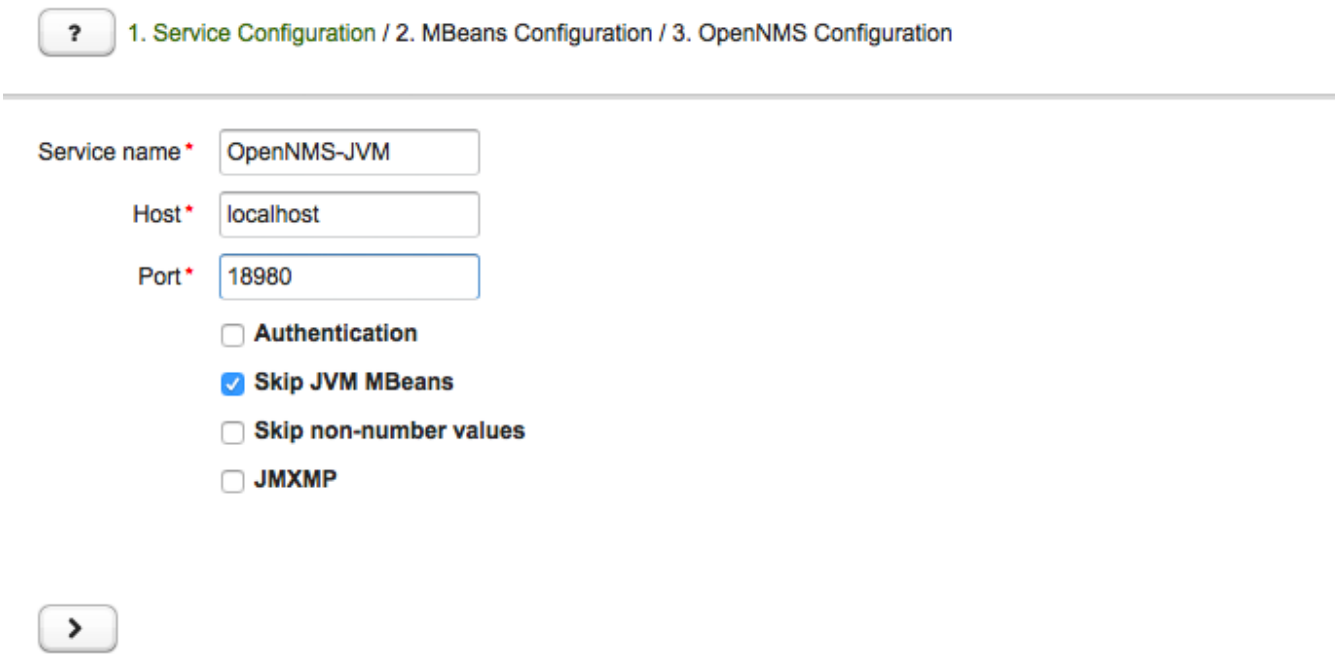


Figure 6. JMX connection configuration window

- *Service name*: The name of the service to bind the *JMX* data collection for *Collectd*
- *Host*: IP address or *FQDN* connecting to the *MBean Server* to load *MBeans* and *Composite Data* into the generation tool
- *Port*: Port to connect to the *MBean Server*
- *Authentication*: Enable / Disable authentication for *JMX* connection with username and password
- *Skip non-number values*: Skip attributes with non-number values
- *JMXMP*: Enable / Disable *JMX Messaging Protocol* instead of using *JMX* over *RMI*

By clicking the arrow (>) the *MBeans* and *Composite Data* will be retrieved with the given connection settings. The data is loaded into the *MBeans Configuration* screen which allows to select metrics for the data collection configuration.

Select MBeans and Composite

The *MBeans Configuration* section is used to assign the *MBean* and *Composite Data attributes* to *RRD domain* specific data types and data source names.

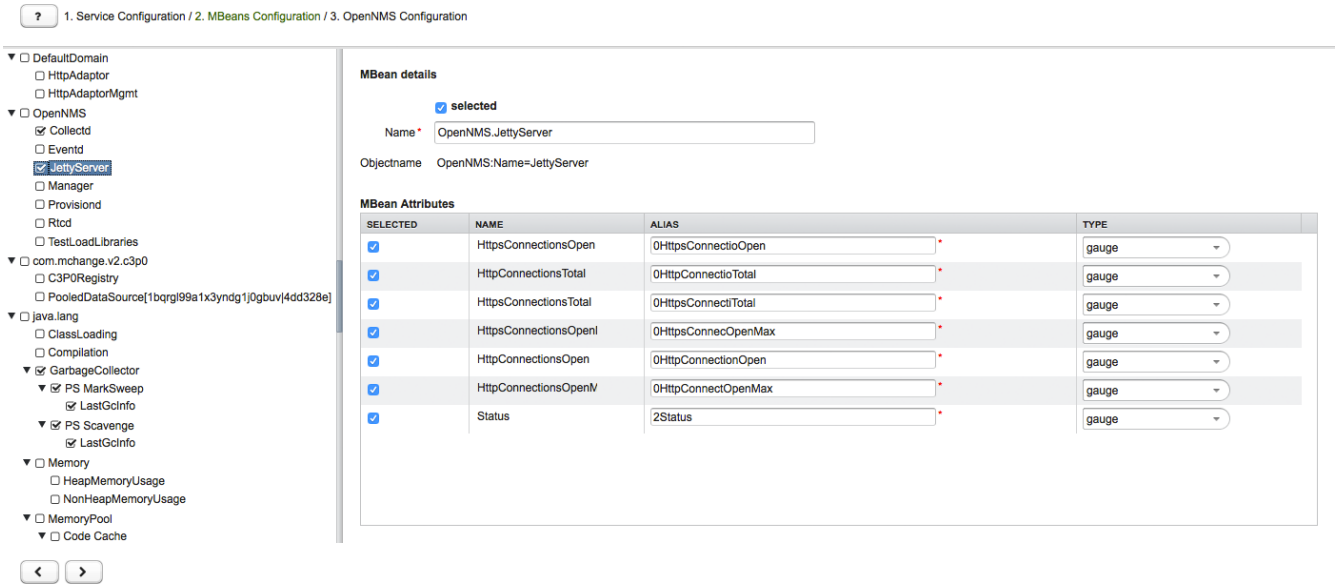


Figure 7. Select MBeans or Composite Data for OpenNMS Horizon data collection

The left sidebar shows the tree with the *JMX Domain*, *MBeans* and *Composite Data* hierarchy retrieved from the *MBean Server*. To select or deselect all attributes use *Mouse right click → select/deselect*.

The right panel shows the *MBean Attributes* with the *RRD* specific mapping and allows to select or deselect specific *MBean Attriubtes* or *Composite Data Attributes* for the data collection configuration.

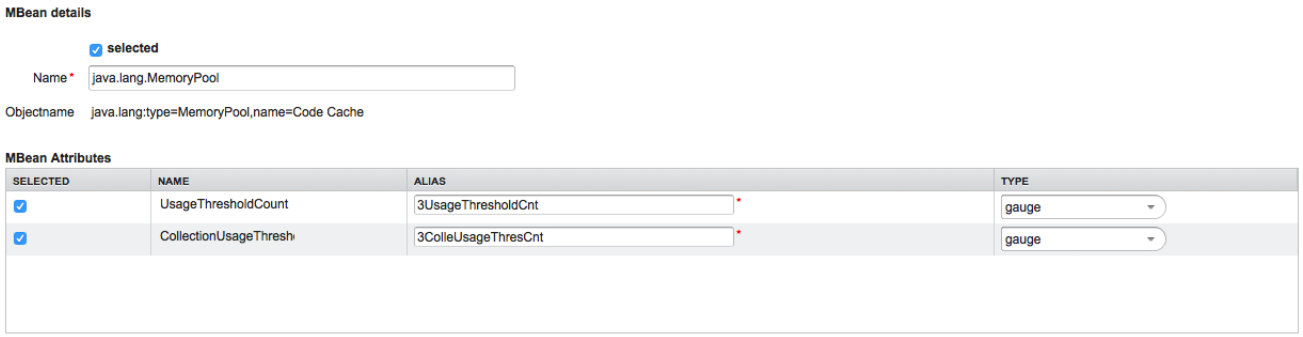


Figure 8. Configure MBean attributes for data collection configuration

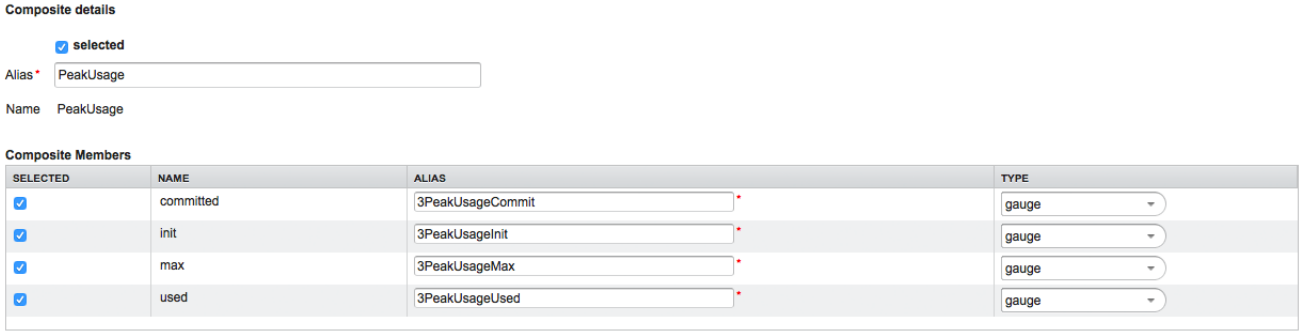


Figure 9. Configure Composite attributes for data collection configuration

- *MBean Name* or *Composite Alias*: Identifies the *MBean* or the *Composite Data* object
- *Selected*: Enable/Disable the *MBean attribute* or *Composite Member* to be included in the data collection configuration
- *Name*: Name of the *MBean attribute* or *Composite Member*
- *Alias*: the *data source name* for persisting measurements in *RRD* or *JRobin* file
- *Type*: *Gauge* or *Counter* data type for persisting measurements in *RRD* or *JRobin* file

The *MBean Name*, *Composite Alias* and *Name* are validated against special characters. For the *Alias* inputs are validated to be not longer then 19 characters and have to be unique in the data collection configuration.

Download and include configuration

The last step is generating the following configuration files for *OpenNMS Horizon*:

- *collectd-configuration.xml*: Generated sample configuration assigned to a service with a matching data collection group
- *jmx-datacollection-config.xml*: Generated *JMX* data collection configuration with the selected *MBeans* and *Composite Data*
- *snmp-graph.properties*: Generated default *RRD* graph definition files for all selected metrics

The content of the configuration files can be copy & pasted or can be downloaded as *ZIP archive*.

If the content of the configuration file exceeds 2,500 lines, the files can only be downloaded as *ZIP archive*.

2.3.2. CLI based utility

The command line *(CLI)* based tool is not installed by default. It is available as *Debian* and *RPM* package in the official repositories.

Installation

RHEL based installation with Yum

```
yum install opennms-jmx-config-generator
```

Debian based installation with apt

```
apt-get install opennms-jmx-config-generator
```

Installation from source

It is required to have the *Java 8 Development Kit* with *Apache Maven* installed. The `mvn` binary has to be in the path environment. After cloning the repository you have to enter the source folder and compile an executable *JAR*.

```
cd opennms/features/jmx-config-generator
mvn package
```

Inside the newly created `target` folder a file named `jmxconfiggenerator-<VERSION>-onejar.jar` is present. This file can be invoked by:

```
java -jar target/jmxconfiggenerator-18.0.0-SNAPSHOT-onejar.jar
```

Usage

After installing the the *JMX Config Generator* the tool's wrapper script is located in the `${OPENNMS_HOME}/bin` directory.

```
$ cd /path/to/opennms/bin
$ ./jmx-config-generator
```

When invoked without parameters the usage and help information is printed.

The *JMX Config Generator* uses sub-commands for the different configuration generation tasks. Each of these sub-commands provide different options and parameters. The command line tool accepts the following sub-commands.

Sub-command	Description
`query`	Queries a *MBean Server* for certain *MBeans* and *attributes*.
`generate-conf`	Generates a valid `jmx-datacollection-config.xml` file.
`generate-graph`	Generates a *RRD* graph definition file with matching graph definitions for a given `jmx-datacollection-config.xml`.

The following global options are available in each of the sub-commands of the tool:

Option/Argument	Description	Default
`-h (--help)`	Show help and usage information.	false
`-v (--verbose)`	Enables verbose mode for debugging purposes.	false

Sub-command: query

This sub-command is used to query a *MBean Server* for it's available *MBean* objects. The following example queries the server `myserver` with the credentials `myusername/mypassword` on port `7199` for *MBean objects* in the `java.lang` domain.

```
./jmx-config-generator query --host myserver --username myusername --password
mypassword --port 7199 "java.lang:*"
java.lang:type=ClassLoading
    description: Information on the management interface of the MBean
    class name: sun.management.ClassLoadingImpl
    attributes: (5/5)
        TotalLoadedClassCount
            id: java.lang:type=ClassLoading:TotalLoadedClassCount
            description: TotalLoadedClassCount
            type: long
            isReadable: true
            isWritable: false
            isIs: false
        LoadedClassCount
            id: java.lang:type=ClassLoading:LoadedClassCount
            description: LoadedClassCount
            type: int
            isReadable: true
            isWritable: false
            isIs: false

<output omitted>
```

The following command line options are available for the *query* sub-command.

Option/Argument	Description	Default
`<filter criteria>`	A filter criteria to query the *MBean Server* for. The format is `<objectname>[:attribute name]`. The `<objectname>` accepts the default *JMX* object name pattern to identify the *MBeans* to be retrieved. If `null` all domains are shown. If no key properties are specified, the domain's *MBeans* are retrieved. To execute for certain attributes, you have to add `:<attribute name>`. The `<attribute name>` accepts regular expressions. When multiple `<filter criteria>` are provided they are `OR` concatenated.	-
`--host <host>`	Hostname or IP address of the remote *JMX* host.	-
`--ids-only`	Only show the ids of the attributes.	false
`--ignore <filter criteria>`	Set `<filter criteria>` to ignore while running.	-
`--include-values`	Include attribute values.	false
`--jmxmp`	Use *JMXMP* and not *JMX over RMI*.	false
`--password <password>`	Password for *JMX* authentication.	-
`--port <port>`	Port of *JMX* service.	-

Option/Argument	Description	Default
`--show-domains`	Only lists the available domains.	true
`--show-empty`	Includes *MBeans*, even if they do not have attributes. Either due to the `<filter criteria>` or while there are none.	false

`--url <url>`	Custom connection *URL*
	`<hostname>:<port>`
	`service:jmx:<protocol>:<sap>`
	`service:jmx:remoting-jmx://<hostname>:<port>`
		-

`--username <username>`	Username for *JMX* authentication.	-
`-h (--help)`	Show help and usage information.	false
`-v (--verbose)`	Enables verbose mode for debugging purposes.	false

Sub-command: generate-conf

This sub-command can be used to generate a valid `jmx-datacollection-config.xml` for a given set of *MBean objects* queried from a *MBean Server*.

The following example generate a configuration file `myconfig.xml` for *MBean* objects in the `java.lang` domain of the server `myserver` on port `7199` with the credentials `myusername/mypassword`. You have to define either an *URL* or a hostname and port to connect to a *JMX* server.

```
jmx-config-generator generate-conf --host myserver --username myusername --password
mypassword --port 7199 "java.lang:*" --output myconfig.xml
Dictionary entries loaded: '18'
```

The following options are available for the *generate-conf* sub-command.

Option/Argument	Description	Default
`<attribute id>`	A list of attribute Ids to be included for the generation of the configuration file.	-
`--dictionary <file>`	Path to a dictionary file for replacing attribute names and part of *MBean* attributes. The file should have for each line a replacement, e.g. Auxillary:Auxil.	-
`--host <host>`	Hostname or IP address of *JMX* host.	-
`--jmxmp`	Use *JMXMP* and not *JMX over RMI*.	false
`--output <file>`	Output filename to write generated `jmx-datacollection-config.xml`.	-
`--password <password>`	Password for *JMX* authentication.	-
`--port <port>`	Port of *JMX* service	-

Option/Argument	Description	Default
`--print-dictionary`	Prints the used dictionary to *STDOUT*. May be used with `--dictionary`	false
`--service <value>`	The *Service Name* used as *JMX* data collection name.	anyservice
`--skipDefaultVM`	Skip default JavaVM Beans.	false
`--skipNonNumber`	Skip attributes with non-number values	false

`--url <url>`	Custom connection *URL*
	`<hostname>:<port>`
	`service:jmx:<protocol>:<sap>`
	`service:jmx:remoting-jmx://<hostname>:<port>`
		-

`--username <username>`	Username for *JMX* authentication	-
`-h (--help)`	Show help and usage information.	false
`-v (--verbose)`	Enables verbose mode for debugging purposes.	false

The option `--skipDefaultVM` offers the ability to ignore the *MBeans* provided as standard by the *JVM* and just create configurations for the *MBeans* provided by the *Java Application* itself. This is particularly useful if an optimized configuration for the JVM already exists. If the `--skipDefaultVM` option is not set the generated configuration will include the MBeans of the JVM and the MBeans of the Java Application.

Check the file and see if there are `alias` names with more than 19 characters. This errors are marked with `NAME_CRASH_AS_19_CHAR_VALUE`

Sub-command: generate-graph

This sub-command generates a *RRD* graph definition file for a given configuration file. The following example generates a graph definition file `mygraph.properties` using the configuration in file `myconfig.xml`.

```
./jmx-config-generator generate-graph --input myconfig.xml --output mygraph.properties
reports=java.lang.ClassLoading.MBeanReport, \
java.lang.ClassLoading.0TotalLoadeClassCnt.AttributeReport, \
java.lang.ClassLoading.0LoadedClassCnt.AttributeReport, \
java.lang.ClassLoading.0UnloadedClassCnt.AttributeReport, \
java.lang.Compilation.MBeanReport, \
<output omitted>
```

The following options are available for this sub-command.

Option/Argument	Description	Default
`--input <jmx-datacollection.xml>`	Configuration file to use as input to generate the graph properties file	-
`--output <file>`	Output filename for the generated graph properties file.	-
`--print-template`	Prints the default template.	false
`--template <file>`	Template file using *Apache Velocity* template engine to be used to generate the graph properties.	-
`-h (--help)`	Show help and usage information.	false
`-v (--verbose)`	Enables verbose mode for debugging purposes.	false

Graph Templates

The *JMX Config Generator* uses a template file to generate the graphs. It is possible to use a user-defined template. The option `--template` followed by a file lets the *JMX Config Generator* use the external template file as base for the graph generation. The following example illustrates how a custom template `mytemplate.vm` is used to generate the graph definition file `mygraph.properties` using the configuration in file `myconfig.xml`.

```
./jmx-config-generator generate-graph --input myconfig.xml --output mygraph.properties
--template mytemplate.vm
```

The template file has to be an *Apache Velocity* template. The following sample represents the template that is used by default:

```
reports=#foreach( $report in $reportsList )
${report.id}#if( $foreach.hasNext ), \
#end
#end

#foreach( $report in $reportsBody )

#[[##########################################]]#
#[[##]]# $report.id
#[[##########################################]]#
report.${report.id}.name=${report.name}
report.${report.id}.columns=${report.graphResources}
report.${report.id}.type=interfaceSnmp
report.${report.id}.command=--title="${report.title}" \
 --vertical-label="${report.verticalLabel}" \
#foreach($graph in $report.graphs )
 DEF:${graph.id}={rrd${foreach.count}}:${graph.resourceName}:AVERAGE \
 AREA:${graph.id}#${graph.coloreB} \
 LINE2:${graph.id}#${graph.coloreA}:"${graph.description}" \
 GPRINT:${graph.id}:AVERAGE:" Avg \\: %8.2lf %s" \
 GPRINT:${graph.id}:MIN:" Min \\: %8.2lf %s" \
 GPRINT:${graph.id}:MAX:" Max \\: %8.2lf %s\\n" \
#end

#end
```

The *JMX Config Generator* generates different types of graphs from the `jmx-datacollection-config.xml`. The different types are listed below:

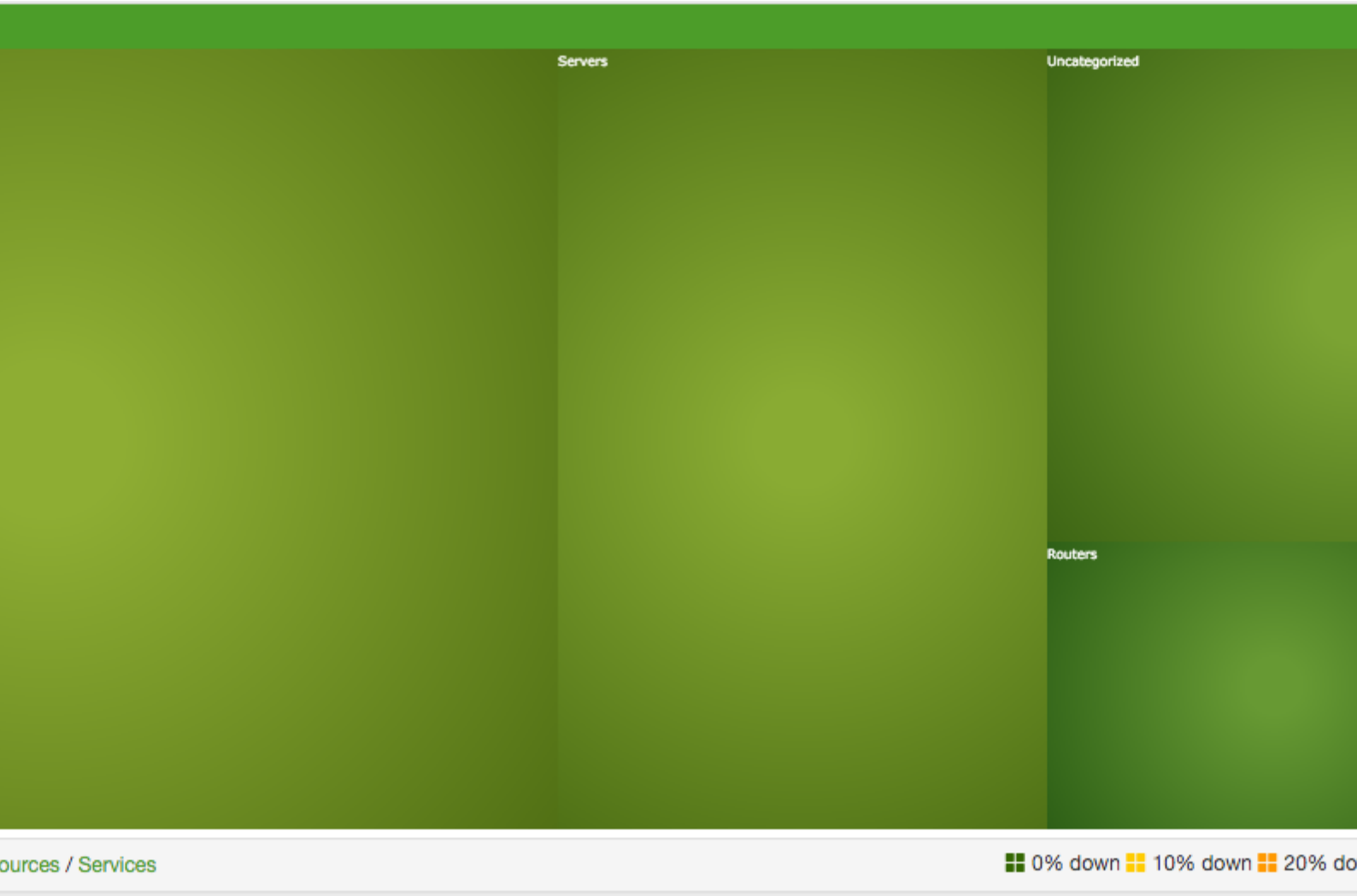
Type	Description
AttributeReport	For each attribute of any *MBean* a graph will be generated. Composite attributes will be ignored.
MbeanReport	For each *MBean* a combined graph with all attributes of the *MBeans* is generated. Composite attributes will be ignored.
CompositeReport	For each composite attribute of every *MBean* a graph is generated.
CompositeAttributeReport	For each composite member of every *MBean* a combined graph with all composite attributes is generated.

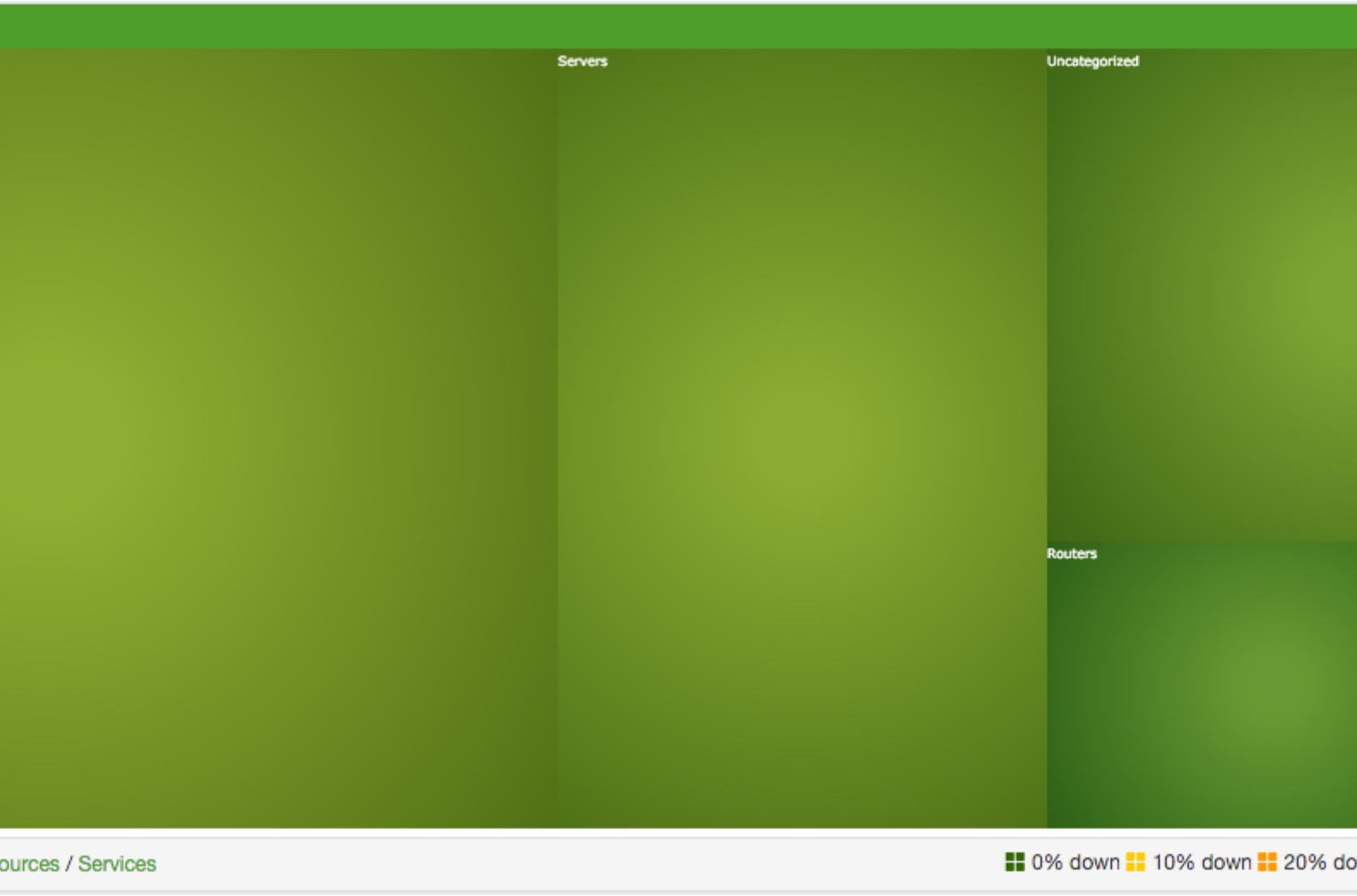
2.4. Heatmap

The *Heatmap* can be either be used to display unacknowledged alarms or to display ongoing outages of nodes. Each of this visualizations can be applied on categories, foreign sources or services of nodes. The sizing of an entity is calculated by counting the services inside the entity. Thus, a node with fewer services will appear in a smaller box than a node with more services.

The feature is by default deactivated and is configured through `opennms.properties`.

Heatmap visualizations of alarms





ources / Services

0% down 10% down 20% do

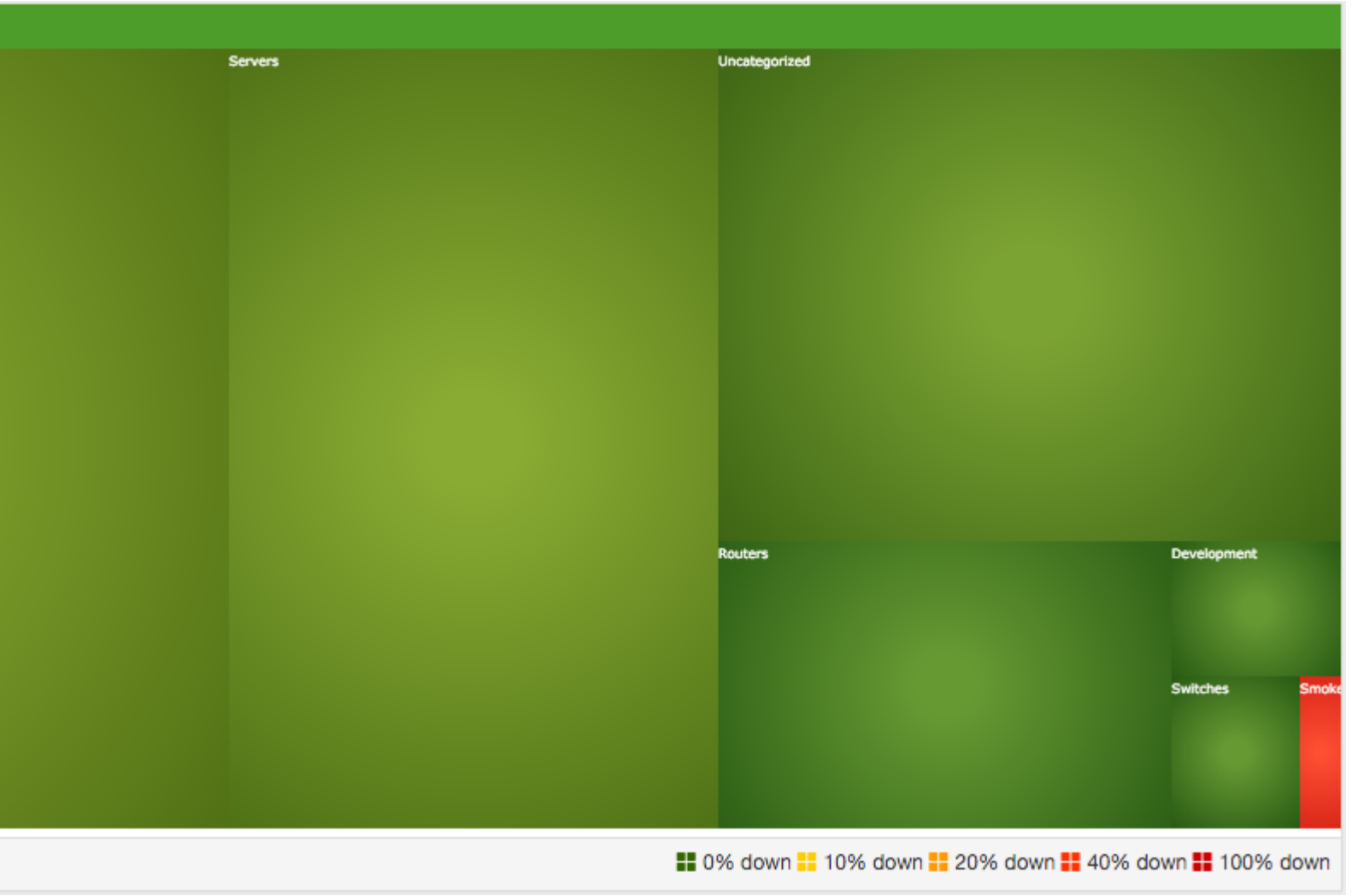


Table 5. Grafana Dashboard configuration properties

Name	Type	Description	Default
`org.opennms.heatmap.defaultMode`	*String*	There exist two options for using the heatmap: `alarms` and `outages`. This option configures which are displayed per default.	`alarms`
`org.opennms.heatmap.defaultHeatmap`	*String*	This option defines which *Heatmap* is displayed by default. Valid options are `categories`, `foreignSources` and `monitoredServices`.	`categories`
`org.opennms.heatmap.categoryFilter`	*String*	The following option is used to filter for categories to be displayed in the *Heatmap*. This option uses the Java regular expression syntax. The default is `.*` so all categories will be displayed.	`.*`

Name	Type	Description	Default
`org.opennms.heatmap.foreignSourceFilter`	*String*	The following option is used to filter for foreign sources to be displayed in the *Heatmap*. This option uses the Java regular expression syntax. The default is `.*` so all foreign sources will be displayed.	`.*`
`org.opennms.heatmap.serviceFilter`	*String*	The following option is used to filter for services to be displayed in the *Heatmap*. This option uses the Java regular expression syntax. The default is `.*` so all services will be displayed.	`.*`
`org.opennms.heatmap.onlyUnacknowledged`	*Boolean*	This option configures whether only unacknowledged alarms will be taken into account when generating the alarm-based version of the *Heatmap*.	`false`
`org.opennms.web.console.centerUrl`	*String*	You can also place the *Heatmap* on the landing page by setting this option to `/heatmap/heatmap-box.jsp`.	`/surveillance-box.jsp`

You can use negative lookahead expressions for excluding categories you wish not to be displayed in the heatmap, e.g. by using an expression like `^(?!XY).*` you can filter out entities with names starting with `XY`.

Chapter 3. Service Assurance

In *OpenNMS* the daemon to measures service availability and latency is done by *Pollerd*. To run these tests *Service Monitors* are scheduled and run in parallel in a *Thread Pool*. The behavior of *Pollerd* uses the following files for configuration and logging. Functionalities and general concepts are described in the *User Documentation* of *OpenNMS*. This section describes how to configure *Pollerd* for service assurance with all available *Service Monitors* coming with *OpenNMS*.

3.1. Pollerd Configuration

Table 6. Configuration and log files related to Pollerd.

File	Description
`$OPENNMS_HOME/etc/poller-configuration.xml`	Configuration file for monitors and global daemon configuration
`$OPENNMS_HOME/logs/poller.log`	Log file for all monitors and the global *Pollerd*
`$OPENNMS_HOME/etc/response-graph.properties`	RRD graph definitions for service response time measurements
`$OPENNMS_HOME/etc/events/opennms.events.xml`	Event definitions for *Pollerd*, i.e. *nodeLostService*, *interfaceDown* or *nodeDown*

To change the behavior for service monitoring, the `poller-configuration.xml` can be modified. The configuration file is structured in the following parts:

- *Global daemon config*: Define the size of the used *Thread Pool* to run *Service Monitors* in parallel. Define and configure the *Critical Service* for *Node Event Correlation*.
- *Polling packages*: Package to allow grouping of configuration parameters for *Service Monitors*.
- *Downtime Model*: Configure the behavior of *Pollerd* to run tests in case of an *Outage* is detected.
- *Monitor service association*: Based on the name of the service, the implementation for application or network management protocols are assigned.

Global configuration parameters for Pollerd

```
<poller-configuration threads="30" ①
                      pathOutageEnabled="false" ②
                      serviceUnresponsiveEnabled="false"> ③
```

① Size of the *Thread Pool* to run *Service Monitors* in parallel

② Enable or Disable *Path Outage* functionality based on a *Critical Node* in a network path

③ In case of unresponsive service services a *serviceUnresponsive* event is generated and not an outage. It prevents to apply the *Downtime Model* to retest the service after 30 seconds and prevents false alarms.

Configuration changes are applied by restarting *OpenNMS* and *Pollerd*. It is also possible to send an *Event* to *Pollerd* reloading the configuration. An *Event* can be sent on the *CLI* or the *Web User*

Interface.

Send configuration reload event on CLI

```
cd $OPENNMS_HOME/bin
./send-event.pl uei.opennms.org/internal/reloadDaemonConfig --parm 'daemonName
Pollerd'
```

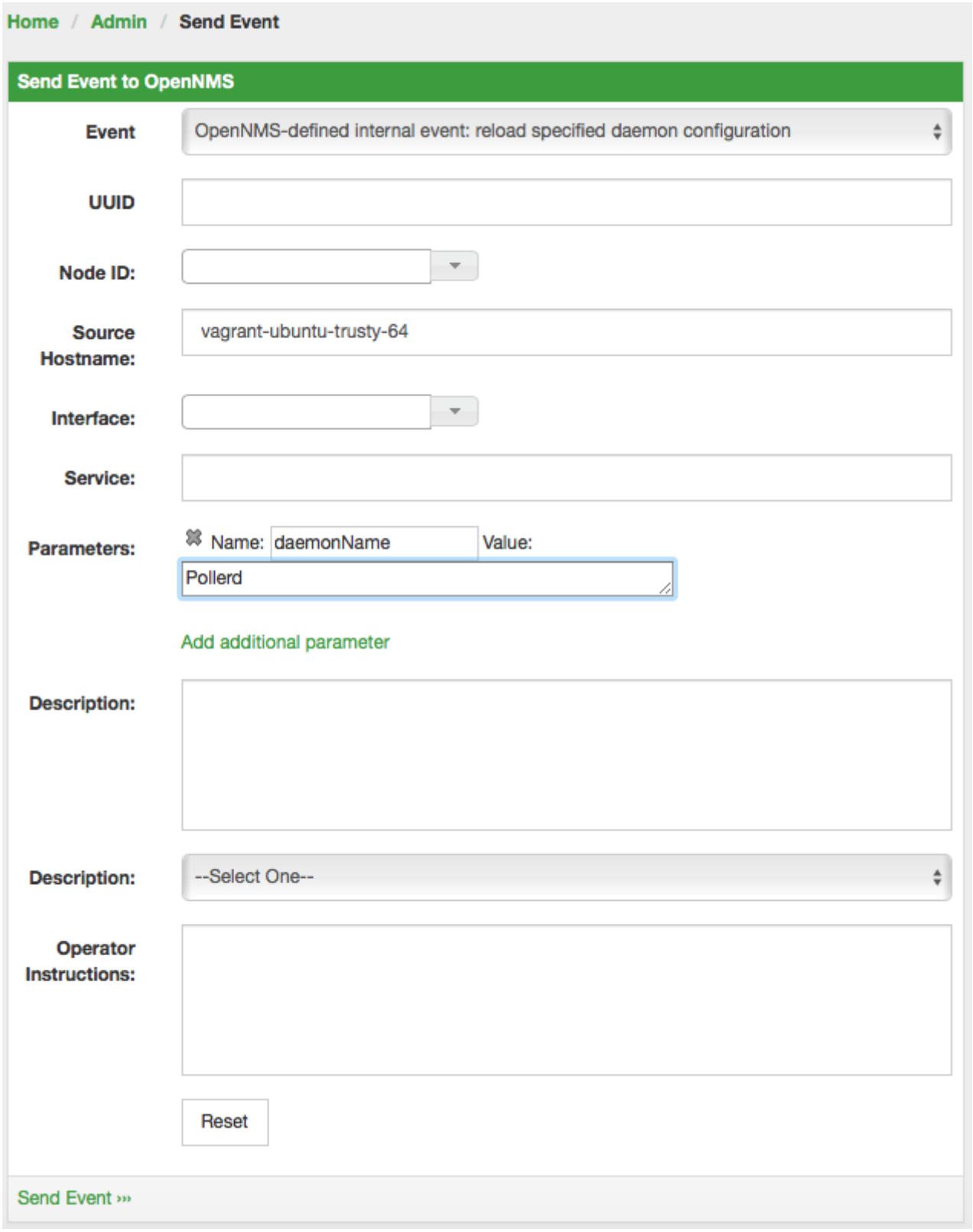


Figure 10. Send configuration reload event with the Web User Interface

If you define **new** services in `poller-configuration.xml` a service restart of *OpenNMS* is necessary.

3.2. Critical Service

The *Critical Service* is used to correlate outages from *Services* to a *nodeDown* or *interfaceDown* event. It is a global configuration of *Pollerd* defined in `poller-configuration.xml`. The *OpenNMS* default configuration enables this behavior.

Critical Service Configuration in Pollerd

```
<poller-configuration threads="30"
                      pathOutageEnabled="false"
                      serviceUnresponsiveEnabled="false">

  <node-outage status="on" ①
               pollAllIfNoCriticalServiceDefined="true"> ②
    <critical-service name="ICMP" /> ③
  </node-outage>
```

① Enable *Node Outage* correlation based on a *Critical Service*

② Optional: In case of nodes without a *Critical Service* this option controls the behavior. If set to `true` then all services will be polled. If set to `false` then the first service in the package that exists on the node will be polled until service is restored, and then polling will resume for all services.

③ Define *Critical Service* for *Node Outage* correlation

3.3. Downtime Model

By default the monitoring interval for a service is 5 minutes. To detect also short services outages, caused for example by automatic network rerouting, the downtime model can be used. On a detected service outage, the interval is reduced to 30 seconds for 5 minutes. If the service comes back within 5 minutes, a shorter outage is documented and the impact on service availability can be less than 5 minutes. This behavior is called *Downtime Model* and is configurable.

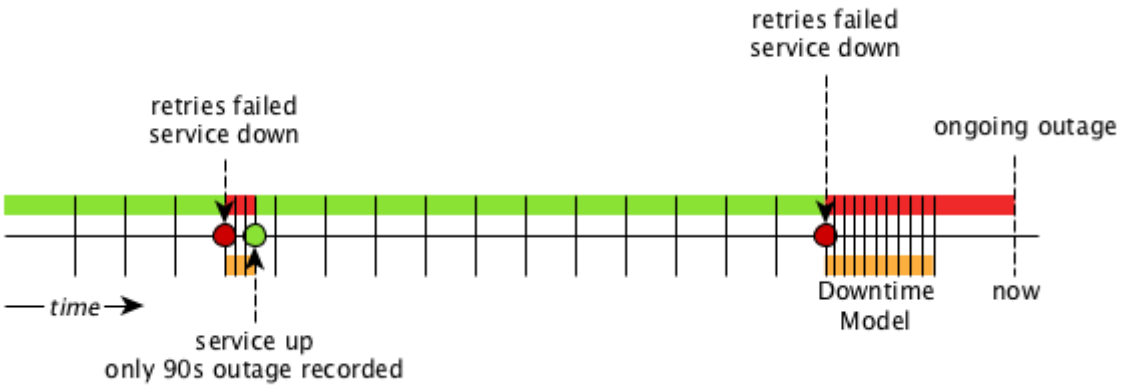


Figure 11. Downtime model with resolved and ongoing outage

In figure Outages and Downtime Model there are two outages. The first outage shows a short outage which was detected as *up* after 90 seconds. The second outage is not resolved now and the monitor has not detected an available service and was not available in the first 5 minutes (10 times 30 second polling). The scheduler changed the polling interval back to 5 minutes.

Example default configuration of the Downtime Model

```
<downtime interval="30000" begin="0" end="300000" /> ①
<downtime interval="300000" begin="300000" end="43200000" /> ②
<downtime interval="600000" begin="43200000" end="432000000" /> ③
<downtime begin="432000000" delete="true" /> ④
```

① from 0 seconds after an outage is detected until 5 minutes the polling interval will be set to 30 seconds

② after 5 minutes of an ongoing outage until 12 hours the polling interval will be set to 5 minutes

③ after 12 hours of an ongoing outage until 5 days the polling interval will be set to 10 minutes

④ after 5 days of an ongoing outage the service will be deleted from the monitoring system

3.4. Path Outages

To reduce the amount of alarms and notifications a *Path Outage* can be configured. This functionality is used to suppress *Notifications* based on the node depending on each other in the network path. The dependency is modeled in the *Node Provisioning* in *Path Outage*.

By default the *Path Outage* feature is disabled and has to be enabled in the `pollerd-configuration.xml`.

It requires the following information:

- *Parent Foreign Source*: The *Foreign Source* where the parent node is defined.
- *Parent Foreign ID*: The *Foreign ID* of the parent *Node* where this node depends on.
- The *IP Interface* selected as *Primary* is used as *Critical IP*

Additionally it is possible to define generic rules for *Path Outages*. For example there is a whole *IP Subnet* behind a *Router* and this *Router* is the *Critical Path* to this *IP Subnet*.

The configuration can be made in *Admin → Configure Notifications → Configure Path Outages*. It requires to specify a *Critical IP* of the *Router* and allows to specify the *IP Subnet* by defining a *Rule/Filter*. They are specified in Rules/Filters in the *OpenNMS Wiki*. In this case, the *Router* with all *Nodes* on the *IP Subnet* are down, but only one *Notification* is sent. All other *Node Down* notifications are suppressed matching the *Rule/Filter* defined in the *Path Outage*.

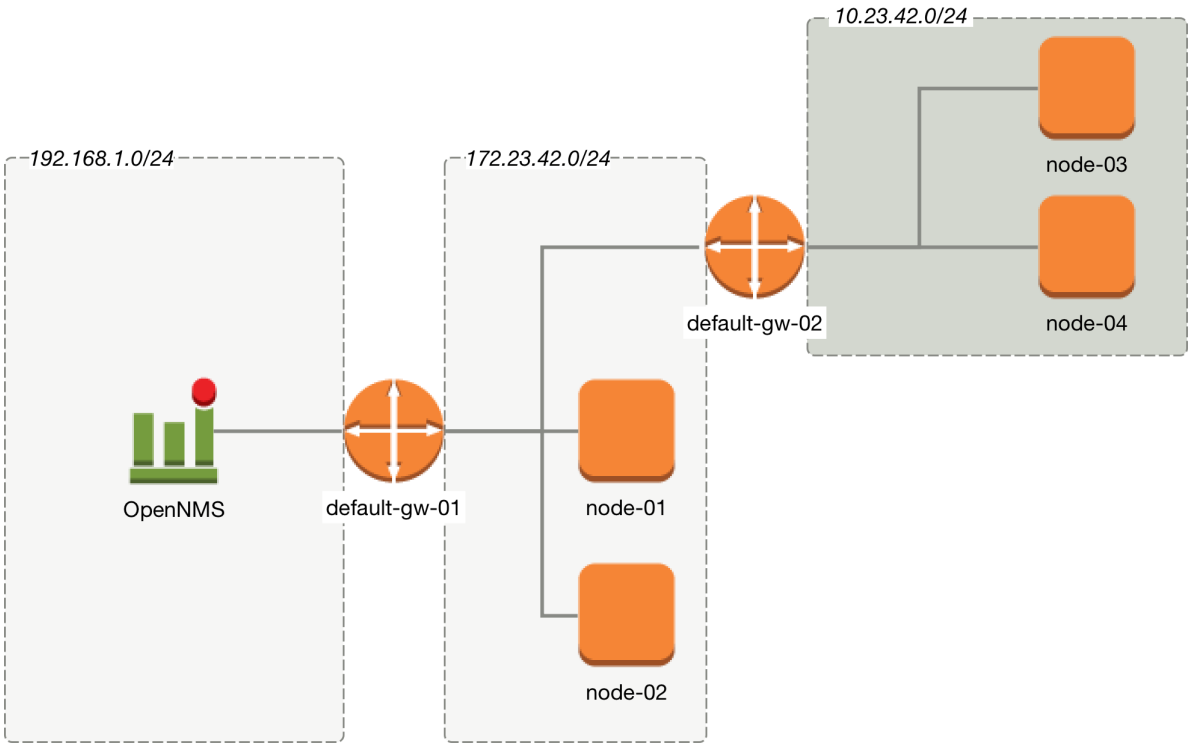


Figure 12. Topology for Path Outage

To configure a *Path Outage* based on the example in figure Topology for Path Outage, the configuration has to be defined as the following.

This example expects all *Nodes* are defined in the same *Foreign Source* named `Network-ACME` and the *Foreign ID* is the same as the *Node Label*.

Table 7. Provisioning for Topology Example

Parent Foreign Source	Parent Foreign ID	Provisioned Node
not defined	*not defined*	`default-gw-01`
`Network-ACME`	`default-gw-01`	`node-01`
`Network-ACME`	`default-gw-01`	`node-02`
`Network-ACME`	`default-gw-01`	`default-gw02`
`Network-ACME`	`default-gw-02`	`node-03`
`Network-ACME`	`default-gw-02`	`node-04`

The *IP Interface* which is set to *Primary* is selected as the *Critical IP*. In this example it is important the *IP interface* on *default-gw-01* in the network *192.168.1.0/24* is set as *Primary* interface. The IP interface in the network *172.23.42.0/24* on *default-gw-02* is set as *Primary* interface.

3.5. Poller Packages

To define more complex monitoring configuration it is possible to group *Service* configurations into

Polling Packages. They allow to define assign to *Nodes* different *Service Configurations*. To assign a *Polling Package* to nodes the Rules/Filters syntax can be used. Each *Polling Package* can have its own Downtime Model configuration.

Multiple packages can be configured, and an interface can exist in more than one package. This gives great flexibility to how the service levels will be determined for a given device.

Polling package assigned to Nodes with Rules and Filters

```
<package name="example1"> ①
  <filter>IPADDR != '0.0.0.0'</filter> ②
  <include-range begin="1.1.1.1" end="254.254.254.254" /> ③
  <include-range begin="::1" end="ffff:ffff:ffff:ffff:ffff:ffff:ffff:ffff" /> ③
```

① Unique name of the polling package.

② Filter can be based on IP address, categories or asset attributes of *Nodes* based on Rules/Filters. The filter is evaluated first and is **required**. This package is used for all *IP Interfaces* which don't have 0.0.0.0 as an assigned *IP address* and is **required**.

③ Allow to specify if the configuration of *Services* is applied on a range of *IP Interfaces* (IPv4 or IPv6).

Instead of the `include-range` it is possible to add one or more specific *IP-Interfaces* with:

Defining a specific IP Interfaces

```
<specific>192.168.1.59</specific>
```

It is also possible to exclude *IP Interfaces* with:

Exclude IP Interfaces

```
<exclude-range begin="192.168.0.100" end="192.168.0.104"/>
```

3.5.1. Response Time Configuration

The definition of *Polling Packages* allows to configure similar services with different polling intervals. All the response time measurements are persisted in *RRD Files* and require a definition. Each *Polling Package* contains a *RRD* definition

RRD configuration for Polling Package example1

```
<package name="example1">
  <filter>IPADDR != '0.0.0.0'</filter>
  <include-range begin="1.1.1.1" end="254.254.254.254" />
  <include-range begin="::1" end="ffff:ffff:ffff:ffff:ffff:ffff:ffff:ffff" />
  <rrd step="300"> ①
    <rra>RRA:AVERAGE:0.5:1:2016</rra> ②
    <rra>RRA:AVERAGE:0.5:12:1488</rra> ③
    <rra>RRA:AVERAGE:0.5:288:366</rra> ④
    <rra>RRA:MAX:0.5:288:366</rra> ⑤
    <rra>RRA:MIN:0.5:288:366</rra> ⑥
</rrd>
```

① Polling interval for all services in this *Polling Package* is reflected in the step of size 300 seconds. All services in this package have to polled in 5 min interval, otherwise response time measurements are not correct persisted.

② 1 step size is persisted 2016 times: 1 * 5 min * 2016 = 7 d, 5 min accuracy for 7 d.

③ 12 steps average persisted 1488 times: 12 * 5 min * 1488 = 62 d, aggregated to 60 min for 62 d.

④ 288 steps average persisted 366 times: 288 * 5 min * 366 = 366 d, aggregated to 24 h for 366 d.

⑤ 288 steps maximum from 24 h persisted for 366 d.

⑥ 288 steps minimum from 24 h persisted for 366 d.

The *RRD* configuration and the service polling interval has to be aligned. In other cases the persisted response time data is not correct displayed in the response time graph.

If the polling interval is changed afterwards, existing *RRD* files needs to be recreated with the new definitions.

3.5.2. Overlapping Services

With the possibility of specifying multiple *Polling Packages* it is possible to use the same *Service* like *ICMP* multiple times. The order how *Polling Packages* in the `poller-configuration.xml` are defined is important when *IP Interfaces* match multiple *Polling Packages* with the same *Service* configuration.

The following example shows which configuration is applied for a specific service:

Overwriting

```
<package name="less-specific">
  <filter>IPADDR != '0.0.0.0'</filter>
  <include-range begin="1.1.1.1" end="254.254.254.254" />
  <include-range begin="::1" end="ffff:ffff:ffff:ffff:ffff:ffff:ffff:ffff" />
  <rrd step="300"> ①
    <rra>RRA:AVERAGE:0.5:1:2016</rra>
    <rra>RRA:AVERAGE:0.5:12:1488</rra>
    <rra>RRA:AVERAGE:0.5:288:366</rra>
    <rra>RRA:MAX:0.5:288:366</rra>
    <rra>RRA:MIN:0.5:288:366</rra>
  </rrd>
  <service name="ICMP" interval="300000" user-defined="false" status="on"> ②
    <parameter key="retry" value="5" /> ③
    <parameter key="timeout" value="10000" /> ④
    <parameter key="rrd-repository" value="/var/lib/opennms/rrd/response" />
    <parameter key="rrd-base-name" value="icmp" />
    <parameter key="ds-name" value="icmp" />
  </service>
  <downtime interval="30000" begin="0" end="300000" />
  <downtime interval="300000" begin="300000" end="43200000" />
  <downtime interval="600000" begin="43200000" end="432000000" />
</package>

<package name="more-specific">
  <filter>IPADDR != '0.0.0.0'</filter>
  <include-range begin="192.168.1.1" end="192.168.1.254" />
  <include-range begin="2600::1" end="2600:::ffff" />
  <rrd step="30"> ①
    <rra>RRA:AVERAGE:0.5:1:20160</rra>
    <rra>RRA:AVERAGE:0.5:12:14880</rra>
    <rra>RRA:AVERAGE:0.5:288:3660</rra>
    <rra>RRA:MAX:0.5:288:3660</rra>
    <rra>RRA:MIN:0.5:288:3660</rra>
  </rrd>
  <service name="ICMP" interval="30000" user-defined="false" status="on"> ②
    <parameter key="retry" value="2" /> ③
    <parameter key="timeout" value="3000" /> ④
    <parameter key="rrd-repository" value="/var/lib/opennms/rrd/response" />
    <parameter key="rrd-base-name" value="icmp" />
    <parameter key="ds-name" value="icmp" />
  </service>
  <downtime interval="10000" begin="0" end="300000" />
  <downtime interval="300000" begin="300000" end="43200000" />
  <downtime interval="600000" begin="43200000" end="432000000" />
</package>
```

① Polling interval in the packages are 300 seconds and 30 seconds

② Different polling interval for the service *ICMP*

③ Different retry settings for the service *ICMP*

④ Different timeout settings for the service *ICMP*

The last *Polling Package* on the service will be applied. This can be used to define a less specific catch all filter for a default configuration. A more specific *Polling Package* can be used to overwrite the default setting. In the example above all *IP Interfaces* in *192.168.1/24* or *2600:/64* will be monitored with ICMP with different polling, retry and timeout settings.

Which *Polling Packages* are applied to the *IP Interface* and *Service* can be found in the *Web User Interface*. The *IP Interface* and *Service* page show which *Polling Package* and *Service* configuration is applied for this specific service.

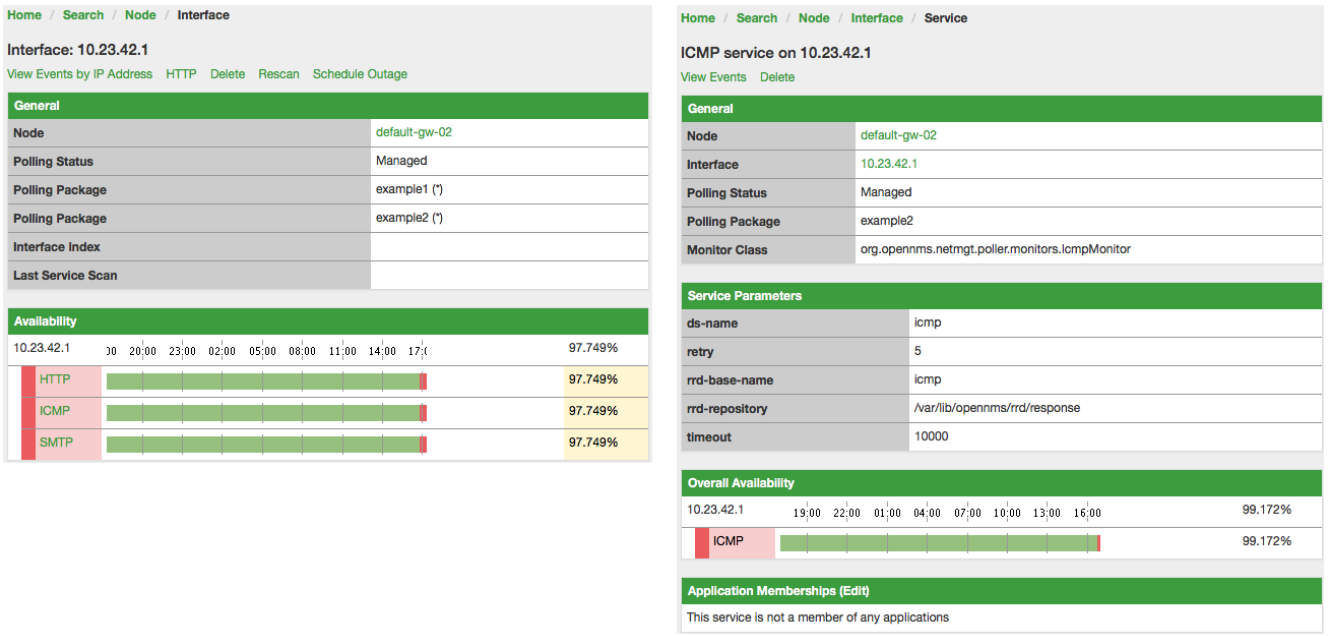


Figure 13. Polling Package applied to IP interface and Service

3.5.3. Test Services on manually

For troubleshooting it is possible to run a test on the *Command Line Interface*.

```
cd $OPENNMS_HOME
./poller-test

-c,--class <arg>       Monitor Class
-i,--ipaddress <arg>   IP Address to test [required]
-P,--package <arg>     Poller Package
-p,--param <arg>       Service parameter ~ key=value
-s,--service <arg>     Service name [required]
```

The following example runs the *ICMP* monitor on a specific *IP Interface*.

Run ICMP monitor configuration defined in specific Polling Package

```
cd $OPENNMS_HOME
./poller-test -i 10.23.42.1 -s ICMP -P example1
```

The output is verbose which allows debugging of *Monitor* configurations. Important output lines are shown as the following:

Important output testing a service on the CLI

```
Checking service ICMP on IP 10.23.42.1 ①
Package: example1 ②
Monitor: org.opennms.netmgt.poller.monitors.IcmpMonitor ③
Parameter ds-name : icmp ④
Parameter rrd-base-name : icmp ④
Parameter rrd-repository : /var/lib/opennms/rrd/response ④
Parameter retry : 2 ⑤
Parameter timeout : 3000 ⑤

Available ? true (status Up[1])
```

① *Service* and *IP Interface* to run the test

② Applied *Service* configuration from *Polling Package* for this test

③ *Service Monitor* used for this test

④ RRD configuration for response time measurement

⑤ Retry and timeout settings for this test

3.6. Service monitors

To support several specific applications and management agents, *Pollerd* executes *Service Monitors*. This section describes all available built-in *Service Monitors* which are available and can be configured to allow complex monitoring. For information how these can be extended, see *Development Guide* of the *OpenNMS* documentation.

3.6.1. AvailabilityMonitor

This monitor tests reachability of a node by using the *isReachable* method of the *InetAddress* java class. The service is considered available if isReachable returns true. See Oracle's documentation for more details.

This monitor is deprecated in favour of the IcmpMonitor monitor. You should only use this monitor on remote pollers running on unusual configurations (See below for more details).

Monitor facts

Class Name	`org.opennms.netmgt.poller.monitors.AvailabilityMonitor`
Remote Enabled	true

Configuration and Usage

Table 8. Monitor specific parameters for the AvailabilityMonitor

Parameter	Description	Required	Default value
`retry`	Number of attempts to have the *isReachable* method return *true*.	optional	`3`
`timeout`	Timeout for the isReachable method, in milliseconds.	optional	`3000`

Examples

```
<service name="AVAIL" interval="300000" user-defined="false" status="on">
  <parameter key="retry" value="2"/>
  <parameter key="timeout" value="5000"/>
</service>

<monitor service="AVAIL" class-name=
"org.opennms.netmgt.poller.monitors.AvailabilityMonitor"/>
```

IcmpMonitor vs AvailabilityMonitor

This monitor has been developped in a time when the IcmpMonitor monitor wasn't remote enabled, to circumvent this limitation. Now, with the JNA ICMP implementation, the IcmpMonitor monitor is remote enabled under most configurations and this monitor shouldn't be needed -unless you're running your remote poller on such an unusual configuration (See also issue NMS-6735 for more information)-.

3.6.2. BgpSessionMonitor

This monitor checks if a BGP-Session to a peering partner (peer-ip) is functional. To monitor the BGP-Session the RFC1269 SNMP MIB is used and test the status of the session using the following OIDs is used:

```
BGP_PEER_STATE_OID = .1.3.6.1.2.1.15.3.1.2.<peer-ip>
BGP_PEER_ADMIN_STATE_OID = .1.3.6.1.2.1.15.3.1.3.<peer-ip>
BGP_PEER_REMOTEAS_OID = .1.3.6.1.2.1.15.3.1.9.<peer-ip>
BGP_PEER_LAST_ERROR_OID = .1.3.6.1.2.1.15.3.1.14.<peer-ip>
BGP_PEER_FSM_EST_TIME_OID = .1.3.6.1.2.1.15.3.1.16.<peer-ip>
```

The `<peer-ip>` is the far end IP address of the BGP session end point.

A SNMP get request for `BGP_PEER_STATE_OID` returns a result between `1` to `6`. The servicestates for OpenNMS Horizon are mapped as follows:

Result	State description	Monitor state in OpenNMS Horizon
1	*Idle*	DOWN
2	*Connect*	DOWN
3	*Active*	DOWN
4	*OpenSent*	DOWN
5	*OpenConfirm*	DOWN
6	*Established*	UP

Monitor facts

Class Name	`org.opennms.netmgt.poller.monitors.BgpSessionMonitor`
Remote Enabled	false

To define the mapping I used the description from RFC1771 BGP Finite State Machine.

Configuration and Usage

Parameter	Description	Required	Default value
`bgpPeerIp`	IP address of the far end BGP peer session	required	-
`retry`	Amount of attempts to get the BGP peer state with SNMP	required	-
`timeout`	Time to wait for the SNMP agents response before trying a next attempt.	required	-

Examples

To monitor the session state *Established* it is necessary to add a service to your poller configuration in '$OPENNMS_HOME/etc/poller-configuration.xml', for example:

```
<!-- Example configuration poller-configuration.xml -->
<service name="BGP-Peer-99.99.99.99-AS65423" interval="300000"
         user-defined="false" status="on">
    <parameter key="retry" value="2" />
    <parameter key="timeout" value="3000" />
    <parameter key="port" value="161" />
    <parameter key="bgpPeerIp" value="99.99.99.99" />
</service>

<monitor service="BGP-Peer-99.99.99.99-AS65423" class-name=
"org.opennms.netmgt.poller.monitors.BgpSessionMonitor" />
```

Error code mapping

The *BGP_PEER_LAST_ERROR_OID* gives an error in HEX-code. To make it human readable a codemapping table is implemented:

Error code	Error Message
0100	Message Header Error
0101	Message Header Error - Connection Not Synchronized
0102	Message Header Error - Bad Message Length
0103	Message Header Error - Bad Message Type
0200	OPEN Message Error
0201	OPEN Message Error - Unsupported Version Number
0202	OPEN Message Error - Bad Peer AS
0203	OPEN Message Error - Bad BGP Identifier
0204	OPEN Message Error - Unsupported Optional Parameter
0205	OPEN Message Error (deprecated)
0206	OPEN Message Error - Unacceptable Hold Time
0300	UPDATE Message Error
0301	UPDATE Message Error - Malformed Attribute List
0302	UPDATE Message Error - Unrecognized Well-known Attribute
0303	UPDATE Message Error - Missing Well-known Attribute
0304	UPDATE Message Error - Attribute Flags Error
0305	UPDATE Message Error - Attribute Length Error
0306	UPDATE Message Error - Invalid ORIGIN Attribute
0307	UPDATE Message Error (deprecated)

Error code	Error Message
0308	UPDATE Message Error - Invalid NEXT_HOP Attribute
0309	UPDATE Message Error - Optional Attribute Error
030A	UPDATE Message Error - Invalid Network Field
030B	UPDATE Message Error - Malformed AS_PATH
0400	Hold Timer Expired
0500	Finite State Machine Error
0600	Cease
0601	Cease - Maximum Number of Prefixes Reached
0602	Cease - Administrative Shutdown
0603	Cease - Peer De-configured
0604	Cease - Administrative Reset
0605	Cease - Connection Rejected
0606	Cease - Other Configuration Change
0607	Cease - Connection Collision Resolution
0608	Cease - Out of Resources

Instead of HEX-Code the error message will be displayed in the service down logmessage. To give some additional informations the logmessage contains also

```
BGP-Peer Adminstate
BGP-Peer Remote AS
BGP-Peer established time in seconds
```

Debugging

If you have problems to detect or monitor the BGP Session you can use the following command to figure out where the problem come from.

```
snmpwalk -v 2c -c <myCommunity> <myRouter2Monitor> .1.3.6.1.2.1.15.3.1.2.99.99.99.99
```

Replace 99.99.99.99 with your BGP-Peer IP. The result should be an Integer between 1 and 6.

3.6.3. BSFMonitor

This monitor runs a *Bean Scripting Framework* BSF compatible script to determine the status of a service. Users can write scripts to perform highly custom service checks. This monitor is not optimised for scale. It's intended for a small number of custom checks or prototyping of monitors.

BSFMonitor vs SystemExecuteMonitor

The *BSFMonitor* avoids the overhead of *fork(2)* that is used by the *SystemExecuteMonitor*. *BSFMonitor* also grants access to a selection of *OpenNMS Horizon* internal methods and classes that can be used in the script.

Monitor facts

Class Name	`org.opennms.netmgt.poller.monitors.BSFMonitor`
Remote Enabled	false

Configuration and Usage

Table 9. Monitor specific parameters for the BSFMonitor

Parameter	Description	Required	Default value
`file-name`	Path to the script file.	required	-
`bsf-engine`	The BSF Engine to run the script in different languages like *Bean Shell*: `bsh.util.BeanShellBSFEngine` *Groovy*: `org.codehaus.groovy.bsf.GroovyEngine` *Jython*: `org.apache.bsf.engines.jython.JythonEngine`	required	-
`run-type`	one of `eval` or `exec`	optional	`eval`
`lang-class`	The BSF language class, like `groovy` or `beanshell`.	optional	file-name extension is interpreted by default
`file-extensions`	comma-separated list	optional	-

Table 10. Beans which can be used in the script

Variable	Type	Description
`map`	*Map<String, Object>*	The *map* contains all various parameters passed to the monitor from the service definition it the `poller-configuration.xml` file.
`ip_addr`	*String*	The IP address that is currently being polled.
`node_id`	*int*	The Node ID of the node the `ip_addr` belongs to.
`node_label`	*String*	The Node Label of the node the `ip_addr` and service belongs to.
`svc_name`	*String*	The name of the service that is being polled.

Variable	Type	Description
`bsf_monitor`	*BSFMonitor*	The instance of the *BSFMonitor* object calling the script. Useful for logging via its log(String sev, String fmt, Object... args) method.
`results`	*HashMap<String, String>*	The script is expected to put its results into this object. The status indication should be set into the entry with key `status`. If the status is not `OK`, a key `reason` should contain a description of the problem.
`times`	*LinkedHashMap<String, Number>*	The script is expected to put one or more response times into this object.

Additionally every parameter added to the service definition in `poller-configuration.xml` is available as a *String* object in the script. The key attribute of the parameter represents the name of the *String* object and the value attribute represents the value of the *String* object.

Please keep in mind, that these parameters are also accessible via the *map* bean.

Avoid non-character names for parameters to avoid problems in the script languages.

Response Codes

The script has to provide a status code that represents the status of the associated service. The following status codes are defined:

Table 11. Status codes

Code	Description
OK	Service is available
UNK	Service status unknown
UNR	Service is unresponsive
NOK	Service is unavailable

Response time tracking

By default the *BSFMonitor* tracks the whole time the script file consumes as the response time. If the response time should be persisted the response time add the following parameters:

RRD response time tracking for this service in `poller-configuration.xml`

```
<!-- where in the filesystem response times are stored -->
<parameter key="rrd-repository" value="/opt/opennms/share/rrd/response" />

<!-- name of the rrd file -->
<parameter key="rrd-base-name" value="minimalbshbase" />

<!-- name of the data source in the rrd file -->
<!-- by default "response-time" is used as ds-name -->
<parameter key="ds-name" value="myResponseTime" />
```

It is also possible to return one or many response times directly from the script. To add custom response times or override the default one, add entries to the *times* object. The entries are keyed with a *String* that names the datasource and have as values a number that represents the response time. To override the default response time datasource add an entry into `times` named `response-time`.

Timeout and Retry

The *BSFMonitor* does not perform any timeout or retry processing on its own. If retry and or timeout behaviour is required, it has to be implemented in the script itself.

Requirements for the script (run-types)

Depending on the `run-type` the script has to provide its results in different ways. For minimal scripts with very simple logic `run-type eval` is the simple option. Scripts running in `eval` mode have to return a *String* matching one of the `status codes`.

If your script is more than a one-liner, `run-type exec` is essentially required. Scripts running in `exec` mode need not return anything, but they have to add a `status` entry with a `status code` to the *results* object. Additionally, the *results* object can also carry a "reason":"message" entry that is used in non `OK` states.

Commonly used language settings

The *BSF* supports many languages, the following table provides the required setup for commonly used languages.

Table 12. BSF language setups

Language	lang-class	bsf-engine	required library
BeanShell	*beanshell*	`bsh.util.BeanShellBSFEngine`	supported by default
Groovy	*groovy*	`org.codehaus.groovy.bsf.GroovyEngine`	groovy-all-[version].jar
Jython	*jython*	`org.apache.bsf.engines.jython.JythonEngine`	jython-[version].jar

Example Bean Shell

BeanShell example `poller-configuration.xml`

```
<service name="MinimalBeanShell" interval="300000" user-defined="true" status="on">
  <parameter key="file-name"  value="/tmp/MinimalBeanShell.bsh"/>
  <parameter key="bsf-engine" value="bsh.util.BeanShellBSFEngine"/>
</service>

<monitor service="MinimalBeanShell" class-name=
"org.opennms.netmgt.poller.monitors.BSFMonitor" />
```

BeanShell example `MinimalBeanShell.bsh` *script file*

```
bsf_monitor.log("ERROR", "Starting MinimalBeanShell.bsf", null);
File testFile = new File("/tmp/TestFile");
if (testFile.exists()) {
  return "OK";
} else {
  results.put("reason", "file does not exist");
  return "NOK";
}
```

Example Groovy

To use the Groovy language an additional library is required. Copy a compatible groovy-all.jar into to `opennms/lib` folder and restart *OpenNMS Horizon*. That makes *Groovy* available for the *BSFMonitor*.

Groovy example `poller-configuration.xml` *with default* `run-type` *set to* `eval`

```
<service name="MinimalGroovy" interval="300000" user-defined="true" status="on">
  <parameter key="file-name"  value="/tmp/MinimalGroovy.groovy"/>
  <parameter key="bsf-engine" value="org.codehaus.groovy.bsf.GroovyEngine"/>
</service>

<monitor service="MinimalGroovy" class-name=
"org.opennms.netmgt.poller.monitors.BSFMonitor" />
```

Groovy example `MinimalGroovy.groovy` *script file for* `run-type eval`

```
bsf_monitor.log("ERROR", "Starting MinimalGroovy.groovy", null);
File testFile = new File("/tmp/TestFile");
if (testFile.exists()) {
  return "OK";
} else {
  results.put("reason", "file does not exist");
  return "NOK";
}
```

Groovy example `poller-configuration.xml` *with* `run-type` *set to* `exec`

```
<service name="MinimalGroovy" interval="300000" user-defined="true" status="on">
  <parameter key="file-name"  value="/tmp/MinimalGroovy.groovy"/>
  <parameter key="bsf-engine" value="org.codehaus.groovy.bsf.GroovyEngine"/>
  <parameter key="run-type" value="exec"/>
</service>

<monitor service="MinimalGroovy" class-name=
"org.opennms.netmgt.poller.monitors.BSFMonitor" />
```

Groovy example `MinimalGroovy.groovy` *script file for* `run-type` *set to* `exec`

```
bsf_monitor.log("ERROR", "Starting MinimalGroovy", null);
def testFile = new File("/tmp/TestFile");
if (testFile.exists()) {
  results.put("status", "OK")
} else {
  results.put("reason", "file does not exist");
  results.put("status", "NOK");
}
```

Example Jython

To use the *Jython* (*Java* implementation of *Python*) language an additional library is required. Copy a compatible `jython-x.y.z.jar` into the `opennms/lib` folder and restart *OpenNMS Horizon*. That makes *Jython* available for the *BSFMonitor*.

Jython example `poller-configuration.xml` *with* `run-type exec`

```
<service name="MinimalJython" interval="300000" user-defined="true" status="on">
  <parameter key="file-name"  value="/tmp/MinimalJython.py"/>
  <parameter key="bsf-engine" value="org.apache.bsf.engines.jython.JythonEngine"/>
  <parameter key="run-type" value="exec"/>
</service>

<monitor service="MinimalJython" class-name=
"org.opennms.netmgt.poller.monitors.BSFMonitor" />
```

Jython example `MinimalJython.py` *script file for* `run-type` *set to* `exec`

```
from java.io import File

bsf_monitor.log("ERROR", "Starting MinimalJython.py", None);
if (File("/tmp/TestFile").exists()):
        results.put("status", "OK")
else:
        results.put("reason", "file does not exist")
        results.put("status", "NOK")
```

We have to use `run-type exec` here because *Jython* chokes on the import keyword in `eval` mode.

As profit that this is really *Python*, notice the substitution of *Python's* None value for Java's null in the log call.

Advanced examples

The following example references all beans that are exposed to the script, including a custom parameter.

Groovy example `poller-configuration.xml`

```
<service name="MinimalGroovy" interval="30000" user-defined="true" status="on">
  <parameter key="file-name"  value="/tmp/MinimalGroovy.groovy"/>
  <parameter key="bsf-engine" value="org.codehaus.groovy.bsf.GroovyEngine"/>

  <!-- custom parameters (passed to the script) -->
  <parameter key="myParameter" value="Hello Groovy" />

  <!-- optional for response time tracking -->
  <parameter key="rrd-repository" value="/opt/opennms/share/rrd/response" />
  <parameter key="rrd-base-name" value="minimalgroovybase" />
  <parameter key="ds-name" value="minimalgroovyds" />
</service>

<monitor service="MinimalGroovy" class-name=
"org.opennms.netmgt.poller.monitors.BSFMonitor" />
```

Groovy example Bean referencing script file

```
bsf_monitor.log("ERROR", "Starting MinimalGroovy", null);

//list of all available objects from the BSFMonitor
Map<String, Object> map = map;
bsf_monitor.log("ERROR", "---- map ----", null);
bsf_monitor.log("ERROR", map.toString(), null);

String ip_addr = ip_addr;
bsf_monitor.log("ERROR", "---- ip_addr ----", null);
bsf_monitor.log("ERROR", ip_addr, null);

int node_id = node_id;
bsf_monitor.log("ERROR", "---- node_id ----", null);
bsf_monitor.log("ERROR", node_id.toString(), null);

String node_label = node_label;
bsf_monitor.log("ERROR", "---- node_label ----", null);
bsf_monitor.log("ERROR", node_label, null);

String svc_name = svc_name;
bsf_monitor.log("ERROR", "---- svc_name ----", null);
bsf_monitor.log("ERROR", svc_name, null);

org.opennms.netmgt.poller.monitors.BSFMonitor bsf_monitor = bsf_monitor;
bsf_monitor.log("ERROR", "---- bsf_monitor ----", null);
bsf_monitor.log("ERROR", bsf_monitor.toString(), null);

HashMap<String, String> results = results;
bsf_monitor.log("ERROR", "---- results ----", null);
bsf_monitor.log("ERROR", results.toString(), null);
```

```
LinkedHashMap<String, Number> times = times;
bsf_monitor.log("ERROR", "---- times ----", null);
bsf_monitor.log("ERROR", times.toString(), null);

// reading a parameter from the service definition
String myParameter = myParameter;
bsf_monitor.log("ERROR", "---- myParameter ----", null);
bsf_monitor.log("ERROR", myParameter, null);

// minimal example
def testFile = new File("/tmp/TestFile");
if (testFile.exists()) {
  bsf_monitor.log("ERROR", "Done MinimalGroovy ---- OK ----", null);
  return "OK";
} else {

  results.put("reason", "file does not exist");
  bsf_monitor.log("ERROR", "Done MinimalGroovy ---- NOK ----", null);
  return "NOK";
}
```

3.6.4. CiscoIpSlaMonitor

This monitor can be used to monitor IP SLA configurations on your Cisco devices. This monitor supports the following SNMP OIDS from CISCO-RTT-MON-MIB:

```
RTT_ADMIN_TAG_OID = .1.3.6.1.4.1.9.9.42.1.2.1.1.3
RTT_OPER_STATE_OID = .1.3.6.1.4.1.9.9.42.1.2.9.1.10
RTT_LATEST_OPERSENSE_OID = .1.3.6.1.4.1.9.9.42.1.2.10.1.2
RTT_ADMIN_THRESH_OID = .1.3.6.1.4.1.9.9.42.1.2.1.1.5
RTT_ADMIN_TYPE_OID = .1.3.6.1.4.1.9.9.42.1.2.1.1.4
RTT_LATEST_OID = .1.3.6.1.4.1.9.9.42.1.2.10.1.1
```

The monitor can be run in two scenarios. The first one tests the *RTT_LATEST_OPERSENSE* which is a sense code for the completion status of the latest RTT operation. If the *RTT_LATEST_OPERSENSE* returns *ok(1)* the service is marked as *up*.

The second scenario is to monitor the configured threshold in the *IP SLA* config. If the *RTT_LATEST_OPERSENSE* returns with *overThreshold(3)* the service is marked *down*.

Monitor facts

Class Name	`org.opennms.netmgt.poller.monitors.CiscoIpSlaMonitor`
Remote Enabled	false

Configuration and Usage

Table 13. Monitor-specific parameters for the CiscoIpSlaMonitor

Parameter	Description	Required	Default value
`retry`	Number of retries to get the information from the SNMP agent before the service is marked as *down*.	optional	from `snmp-config.xml`
`timeout`	Time in milliseconds to wait for the result from the SNMP agent before making the next attempt.	optional	from `snmp-config.xml`
`admin-tag`	The `tag` attribute from your *IP SLA* configuration you want to monitor.	required	`-`
`ignore-thresh`	Boolean indicates if just the status or configured threshold should be monitored.	required	``

Example for HTTP and ICMP echo reply

In this example we configure an IP SLA entry to monitor Google's website with *HTTP GET* from the Cisco device. We use 8.8.8.8 as our DNS resolver. In our example our SLA says we should reach Google's website within 200ms. To advise co-workers that this monitor entry is used for monitoring, I set the owner to *OpenNMS*. The `tag` is used to identify the entry later in the SNMP table for monitoring.

Cisco device configuration for IP SLA instance for HTTP GET

```
ip sla monitor 1
 type http operation get url http://www.google.de name-server 8.8.8.8
 timeout 3000
 threshold 200
 owner OpenNMS
 tag Google Website
ip sla monitor schedule 3 life forever start-time now
```

In the second example we configure a IP SLA to test if the IP address from www.opennms.org is reachable with ICMP from the perspective of the Cisco device. Like the example above we have a threshold and a timeout.

Cisco device configuration for IP SLA instance for ICMP monitoring.

```
ip sla 1
 icmp-echo 64.146.64.212
 timeout 3000
 threshold 150
 owner OpenNMS
 tag OpenNMS Host
ip sla schedule 1 life forever start-time now
```

It´s not possible to reconfigure an IP SLA entry. If you want to change parameters, you have to delete the whole configuration and reconfigure it with your new parameters. Backup your Cisco configuration manually or take a look at RANCID.

To monitor both of the entries the configuration in `poller-configuration.xml` requires two service definition entries:

```
<service name="IP-SLA-WEB-Google" interval="300000"
  user-defined="false" status="on">
    <parameter key="retry" value="2" />
    <parameter key="timeout" value="3000" />
    <parameter key="admin-tag" value="Google Website" />
    <parameter key="ignore-thresh" value="false" /> ①
</service>
<service name="IP-SLA-PING-OpenNMS" interval="300000"
  user-defined="false" status="on">
    <parameter key="retry" value="2" />
    <parameter key="timeout" value="3000" />
    <parameter key="admin-tag" value="OpenNMS Host" />
    <parameter key="ignore-thresh" value="true" /> ②
</service>

<monitor service="IP-SLA-WEB-Google" class-name=
"org.opennms.netmgt.poller.monitors.CiscoIpSlaMonitor" />
<monitor service="IP-SLA-PING-OpenNMS" class-name=
"org.opennms.netmgt.poller.monitors.CiscoIpSlaMonitor" />
```

① Service is *up* if the IP SLA state is *ok(1)*

② Service is *down* if the IP SLA state is *overThreshold(3)*

3.6.5. CiscoPingMibMonitor

This poller monitor's purpose is to create conceptual rows (entries) in the *ciscoPingTable* on *Cisco IOS* devices that support the CISCO-PING-MIB. These entries direct the remote *IOS* device to ping an IPv4 or IPv6 address with a configurable set of parameters. After the *IOS* device has completed the requested ping operations, the poller monitor queries the *IOS* device to determine the results. If the results indicate success according to the configured parameters in the service configuration, then the monitored service is reported as available and the results are available for optional time-series (RRD) storage. If the results indicate failure, the monitored service is reported unavailable with a descriptive reason code. If something goes wrong during the setup of the entry or the subsequent querying of its status, the monitored service is reported to be in an *unknown* state.

Unlike most poller monitors, the *CiscoPingMibMonitor* does not interpret the `timeout` and `retries` parameters to determine when a poll attempt has timed out or whether it should be attempted again. The `packet-count` and `packet-timeout` parameters instead service this purpose from the perspective of the remote *IOS* device.

Supported MIB OIDs from CISCO_PING_MIB

```
ciscoPingEntry              1.3.6.1.4.1.9.9.16.1.1.1
ciscoPingSerialNumber       1.3.6.1.4.1.9.9.16.1.1.1.1
ciscoPingProtocol           1.3.6.1.4.1.9.9.16.1.1.1.2
ciscoPingAddress            1.3.6.1.4.1.9.9.16.1.1.1.3
ciscoPingPacketCount        1.3.6.1.4.1.9.9.16.1.1.1.4
ciscoPingPacketSize         1.3.6.1.4.1.9.9.16.1.1.1.5
ciscoPingPacketTimeout      1.3.6.1.4.1.9.9.16.1.1.1.6
ciscoPingDelay              1.3.6.1.4.1.9.9.16.1.1.1.7
ciscoPingTrapOnCompletion   1.3.6.1.4.1.9.9.16.1.1.1.8
ciscoPingSentPackets        1.3.6.1.4.1.9.9.16.1.1.1.9
ciscoPingReceivedPackets    1.3.6.1.4.1.9.9.16.1.1.1.10
ciscoPingMinRtt             1.3.6.1.4.1.9.9.16.1.1.1.11
ciscoPingAvgRtt             1.3.6.1.4.1.9.9.16.1.1.1.12
ciscoPingMaxRtt             1.3.6.1.4.1.9.9.16.1.1.1.13
ciscoPingCompleted          1.3.6.1.4.1.9.9.16.1.1.1.14
ciscoPingEntryOwner         1.3.6.1.4.1.9.9.16.1.1.1.15
ciscoPingEntryStatus        1.3.6.1.4.1.9.9.16.1.1.1.16
ciscoPingVrfName            1.3.6.1.4.1.9.9.16.1.1.1.17
```

Prerequisites

- One or more *Cisco* devices running an *IOS* image of recent vintage; any 12.2 or later image is probably fine. Even very low-end devices appear to support the CISCO-PING-MIB.
- The *IOS* devices that will perform the remote pings must be configured with an *SNMP write community* string whose source address access-list includes the address of the OpenNMS Horizon server and whose MIB view (if any) includes the OID of the *ciscoPingTable*.
- The corresponding *SNMP write community* string must be specified in the `write-community` attribute of either the top-level `<snmp-config>` element of `snmp-config.xml` or a `<definition>` child element that applies to the *SNMP-primary* interface of the *IOS* device(s) that will perform the remote pings.

Scalability concerns

This monitor spends a fair amount of time sleeping while it waits for the remote *IOS* device to complete the requested ping operations. The monitor is pessimistic in calculating the delay between creation of the *ciscoPingTable* entry and its first attempt to retrieve the results of that entry's ping operations — it will always wait at least (`packet-count * (packet-timeout + packet-delay)`) milliseconds before even checking whether the remote pings have completed. It's therefore prone to hogging poller threads if used with large values for the `packet-count`, `packet-timeout`, and/or `packet-delay` parameters. Keep these values as small as practical to avoid tying up poller threads

unnecessarily.

This monitor always uses the current time in whole seconds since the UNIX epoch as the instance identifier of the *ciscoPingTable* entries that it creates. The object that holds this identifier is a signed 32-bit integer type, precluding a finer resolution. It's probably a good idea to mix in the least-significant byte of the millisecond-accurate time as a substitute for that of the whole-second-accurate value to avoid collisions. *IOS* seems to clean up entries in this table within a manner of minutes after their ping operations have completed.

Monitor facts

Class Name	`org.opennms.netmgt.poller.monitors.CiscoPingMibMonitor`
Remote Enabled	false

Configuration and Usage

Table 14. Monitor specific parameters for the CiscoPingMibMonitor

Parameter	Description	Required	Default value
`timeout`	A timeout, in milliseconds, that should override the SNMP timeout specified in `snmp-config.xml`. Do not use without a very good reason to do so.	optional	from `snmp-config.xml`
`retry`	Number of retries to attempt if the initial attempt times out. Overrides the equivalent value from `snmp-config.xml`. Do not use unless really needed.	optional	from `snmp-config.xml`
`version`	SNMP protocol version (1, 2c, or 3) to use for operations performed by this service monitor. Do not use with out a very good reason to do so.	optional	from `snmp-config.xml`
`packet-count`	Number of ping packets that the remote *IOS* device should send.	optional	`5`
`packet-size`	Size, in bytes, of each ping packet that the remote *IOS* device should send.	optional	`100`
`packet-timeout`	Timeout, in milliseconds, of each ping packet sent by the remote *IOS* device.	optional	`2000`
`packet-delay`	Delay, in milliseconds, between ping packets sent by the remote *IOS* device.	optional	`0`
`entry-owner`	String value to set as the value of ciscoPingEntryOwner of entries created for this service.	optional	`OpenNMS CiscoPingMibMonitor`

Parameter	Description	Required	Default value
`vrf-name`	String value to set as the VRF (VLAN) name in whose context the remote *IOS* device should perform the pings for this service.	optional	*empty String*
`proxy-node-id`	Numeric database identifier of the node whose primary SNMP interface should be used as the *proxy* for this service. If specified along with the related `proxy-node-foreign-source`, `proxy-node-foreign-id`, and/or `proxy-ip-addr`, this parameter will be the effective one.	optional	-
`proxy-node-foreign-source` `proxy-node-foreign-id`	`foreign-source` name and `foreign-ID` of the node whose primary SNMP interface should be used as the "proxy" for this service. These two parameters are corequisites. If they appear along with the related `proxy-ip-addr`, these parameters will be the effective ones.	optional	-
`proxy-ip-addr`	IP address of the interface that should be used as the *proxy* for this service. Effective only if none of `proxy-node-id`, `proxy-node-foreign-source`, nor `proxy-node-foreign-id` appears alongside this parameter. A value of `${ipaddr}` will be substituted with the IP address of the interface on which the monitored service appears.	optional	-
`target-ip-addr`	IP address that the remote *IOS* device should ping. A value of `${ipaddr}` will be substituted with the IP address of the interface on which the monitored service appears.	optional	-
`success-percent`	A whole-number percentage of pings that must succeed (from the perspective of the remote *IOS* device) in order for this service to be considered available. As an example, if `packet-count` is left at its default value of `5` but you wish the service to be considered available even if only one of those five pings is successful, then set this parameter's value to `20`.	optional	`100`
`rrd-repository`	Base directory of an RRD repository in which to store this service monitor's response-time samples	optional	-

Parameter	Description	Required	Default value
`ds-name`	Name of the RRD datasource (DS) name in which to store this service monitor's response-time samples; rrd-base-name Base name of the RRD file (minus the `.rrd` or `.jrb` file extension) within the specified rrd-repository path in which this service monitor's response-time samples will be persisted	optional	`-`

This is optional just if you can use variables in the configuration

Table 15. Variables which can be used in the configuration

Variable	Description
`${ipaddr}`	This value will be substituted with the IP address of the interface on which the monitored service appears.

Example: Ping the same non-routable address from all routers of customer Foo

A service provider's client, Foo Corporation, has network service at multiple locations. At each Foo location, a point-of-sale system is statically configured at IPv4 address 192.168.255.1. Foo wants to be notified any time a point-of-sale system becomes unreachable. Using an OpenNMS Horizon remote location monitor is not feasible. All of Foo Corporation's CPE routers must be *Cisco IOS* devices in order to achieve full coverage in this scenario.

One approach to this requirement is to configure all of Foo Corporation's premise routers to be in the surveillance categories Customer_Foo, CPE, and Routers, and to use a filter to create a poller package that applies only to those routers. We will use the special value `${ipaddr}` for the `proxy-ip-addr` parameter so that the remote pings will be provisioned on each Foo CPE router. Since we want each Foo CPE router to ping the same IP address 192.168.255.1, we statically list that value for the `target-ip-addr` address.

```
<package name="ciscoping-foo-pos">
  <filter>catincCustomer_Foo & catincCPE & catincRouters & nodeSysOID LIKE
'.1.3.6.1.4.1.9.%'</filter>
  <include-range begin="0.0.0.0" end="254.254.254.254" />
  <rrd step="300">
    <rra>RRA:AVERAGE:0.5:1:2016</rra>
    <rra>RRA:AVERAGE:0.5:12:1488</rra>
    <rra>RRA:AVERAGE:0.5:288:366</rra>
    <rra>RRA:MAX:0.5:288:366</rra>
    <rra>RRA:MIN:0.5:288:366</rra>
  </rrd>
  <service name="FooPOS" interval="300000" user-defined="false" status="on">
    <parameter key="rrd-repository" value="/opt/opennms/share/rrd/response" />
    <parameter key="rrd-base-name" value="ciscoping" />
    <parameter key="ds-name" value="ciscoping" />
    <parameter key="proxy-ip-addr" value="${ipaddr}" />
    <parameter key="target-ip-addr" value="192.168.255.1" />
  </service>
  <downtime interval="30000" begin="0" end="300000" /><!-- 30s, 0, 5m -->
  <downtime interval="300000" begin="300000" end="43200000" /><!-- 5m, 5m, 12h -->
  <downtime interval="600000" begin="43200000" end="432000000" /><!-- 10m, 12h, 5d -->
  <downtime begin="432000000" delete="true" /><!-- anything after 5 days delete -->
</package>

<monitor service="FooPOS" class-name=
"org.opennms.netmgt.poller.monitors.CiscoPingMibMonitor" />
```

Example: Ping from a single IOS device routable address of each router of customer Bar

A service provider's client, Bar Limited, has network service at multiple locations. While OpenNMS Horizon' world-class service assurance is generally sufficient, Bar also wants to be notified any time a premise router at one of their locations unreachable from the perspective of an *IOS* device in Bar's main data center. Some or all of the Bar Limited CPE routers may be non-Cisco devices in this scenario.

To meet this requirement, our approach is to configure Bar Limited's premise routers to be in the surveillance categories Customer_Bar, CPE, and Routers, and to use a filter to create a poller package that applies only to those routers. This time, though, we will use the special value `${ipaddr}` not in the `proxy-ip-addr` parameter but in the `target-ip-addr` parameter so that the remote pings will be performed for each Bar CPE router. Since we want the same *IOS* device 20.11.5.11 to ping the CPE routers, we statically list that value for the `proxy-ip-addr` address. Example `poller-configuration.xml` additions

```
<package name="ciscoping-bar-cpe">
  <filter>catincCustomer_Bar & catincCPE & catincRouters</filter>
  <include-range begin="0.0.0.0" end="254.254.254.254" />
  <rrd step="300">
    <rra>RRA:AVERAGE:0.5:1:2016</rra>
    <rra>RRA:AVERAGE:0.5:12:1488</rra>
    <rra>RRA:AVERAGE:0.5:288:366</rra>
    <rra>RRA:MAX:0.5:288:366</rra>
    <rra>RRA:MIN:0.5:288:366</rra>
  </rrd>
  <service name="BarCentral" interval="300000" user-defined="false" status="on">
    <parameter key="rrd-repository" value="/opt/opennms/share/rrd/response" />
    <parameter key="rrd-base-name" value="ciscoping" />
    <parameter key="ds-name" value="ciscoping" />
    <parameter key="proxy-ip-addr" value="20.11.5.11" />
    <parameter key="target-ip-addr" value="${ipaddr}" />
  </service>
  <downtime interval="30000" begin="0" end="300000" /><!-- 30s, 0, 5m -->
  <downtime interval="300000" begin="300000" end="43200000" /><!-- 5m, 5m, 12h -->
  <downtime interval="600000" begin="43200000" end="432000000" /><!-- 10m, 12h, 5d -->
  <downtime begin="432000000" delete="true" /><!-- anything after 5 days delete -->
</package>

<monitor service="BarCentral" class-name=
"org.opennms.netmgt.poller.monitors.CiscoPingMibMonitor" />
```

3.6.6. CitrixMonitor

This monitor is used to test if a Citrix® Server or XenApp Server® is providing the *Independent Computing Architecture* (*ICA*) protocol on TCP 1494. The monitor opens a TCP socket and tests the greeting banner returns with `ICA`, otherwise the service is unavailable.

Monitor facts

Class Name	`org.opennms.netmgt.poller.monitors.CitrixMonitor`
Remote Enabled	true

Configuration and Usage

Table 16. Monitor specific parameters for the CitrixMonitor

Parameter	Description	Required	Default value
`retry`	Amount of attempts opening a connection and try to get the greeting banner before the service goes down	optional	`0`
`timeout`	Time to wait retrieving the greeting banner `ICA` from TCP connection before trying a next attempt.	optional	`3000 ms`

Parameter	Description	Required	Default value
port	TCP port where the *ICA* protocol is listening.	optional	1494

If you have configure the *Metaframe Presentation Server Client* using *Session Reliability*, the TCP port is 2598 instead of 1494. You can find additional information on CTX104147. It is not verified if the monitor works in this case.

Examples

The following example configures OpenNMS Horizon to monitor the ICA protocol on TCP 1494 with 2 retries and waiting 5 seconds for each retry.

```
<service name="Citrix-TCP-ICA" interval="300000" user-defined="false" status="on">
    <parameter key="retry" value="2" />
    <parameter key="timeout" value="5000" />
</service>

<monitor service="Citrix-TCP-ICA" class-name=
"org.opennms.netmgt.poller.monitors.CitrixMonitor" />
```

3.6.7. DhcpMonitor

This monitor is used to monitor the availability and functionality of DHCP servers. This monitor has two parts, the first one is the monitor class *DhcpMonitor* executed by *Pollerd* and the second part is a background daemon *Dhcpd* running inside the OpenNMS Horizon JVM and listening for DHCP responses. A DHCP server is tested by sending a *DISCOVER* message. If the DHCP server responds with an *OFFER* the service is marked as up. The *Dhcpd* background daemon is disabled by default and has to be activated in `service-configuration.xml` in OpenNMS Horizon by setting `service enabled="true"`. The behavior for testing the DHCP server can be modified in the `dhcp-configuration.xml` configuration file.

It is required to install the `opennms-plugin-protocol-dhcp` before you can use this feature.

Installing the opennms-plugin-protocol-dhcp package

```
{apt-get,yum} install {opennms-package-base-name}-plugin-protocol-dhcp
```

If you try to start OpenNMS Horizon without the *opennms-plugin-protocol-dhcp* you will see the following error message in `output.log`:

```
An error occurred while attempting to start the "OpenNMS:Name=Dhcpd" service (class
org.opennms.netmgt.dhcpd.jmx.Dhcpd).  Shutting down and exiting.
java.lang.ClassNotFoundException: org.opennms.netmgt.dhcpd.jmx.Dhcpd
```

Make sure no DHCP client is running on the OpenNMS Horizon server and using port UDP/68. If UDP/68 is already in use, you will find an error message in the manager.log. You can test if a process is listening on udp/68 with `sudo ss -lnpu sport = :68`.

Monitor facts

Class Name	`org.opennms.protocols.dhcp.monitor.DhcpMonitor`
Remote Enabled	false

Table 17. Service monitor parameters configured in `poller-configuration.xml`

Parameter	Description	Required	Default value
`retry`	Number of retries before the service is marked as down	optional	`0`
`timeout`	Time in milliseconds to wait for the DHCP response from the server	optional	`3000`
`rrd-repository`	The location to write RRD data. Generally, you will not want to change this from default	optional	`$OPENNMS_HOME/share/rrd/response`
`rrd-base-name`	The name of the RRD file to write (minus the extension, .rrd or .jrb)	optional	`dhcp`
`ds-name`	This is the name as reference for this particular data source in the RRD file	optional	`dhcp`

Dhcpd configuration

Table 18. Dhcpd parameters in `dhcp-configuration.xml`.

Parameter	Description	Required	Default value
`port`	Defines the port your dhcp server is using	required	`5818`
`macAddress`	The MAC address which OpenNMS Horizon uses for a dhcp request	required	`00:06:0D:BE:9C:B2`

`myIpAddress`	This parameter will usually be set to the IP address of the OpenNMS Horizon server, which puts the DHCP poller in `relay` mode as opposed to `broadcast` mode. In `relay` mode, the DHCP server being polled will unicast its responses directly back to the IP address specified by `myIpAddress` rather than broadcasting its responses. This allows DHCP servers to be polled even though they are not on the same subnet as the OpenNMS Horizon server, and without the aid of an external relay. *Usage:* `myIpAddress="10.11.12.13"` or `myIpAddress="broadcast"`	required	`broadcast`
`extendedMode`	When extendedMode is false, the DHCP poller will send a DISCOVER and expect an OFFER in return. When extendedMode is true, the DHCP poller will first send a DISCOVER. If no valid response is received it will send an INFORM. If no valid response is received it will then send a REQUEST. OFFER, ACK, and NAK are all considered valid responses in extendedMode. *Usage:* `extendedMode="true"` or `extendedMode="false"`	required	`false`
`requestIpAddress`	This parameter only applies to REQUEST queries sent to the DHCP server when extendedMode is true. If an IP address is specified, that IP address will be requested in the query. If `targetHost` is specified, the DHCP server's own IP address will be requested. Since a well-managed server will probably not respond to a request for its own IP, this parameter can also be set to `targetSubnet`. This is similar to `targetHost` except the DHCP server's IP address is incremented or decremented by 1 to obtain an ip address that is on the same subnet. (The resulting address will not be on the same subnet if the DHCP server's subnet is a /32 or /31. Otherwise, the algorithm used should be reliable.) *Usage:* `requestIpAddress="10.77.88.99"` or `requestIpAddress="targetHost"` or `requestIpAddress="targetSubnet"`	required	`false`

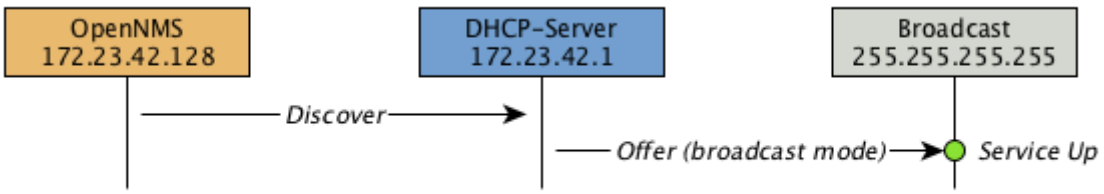


Figure 14. Visualization of DHCP message flow in broadcast mode

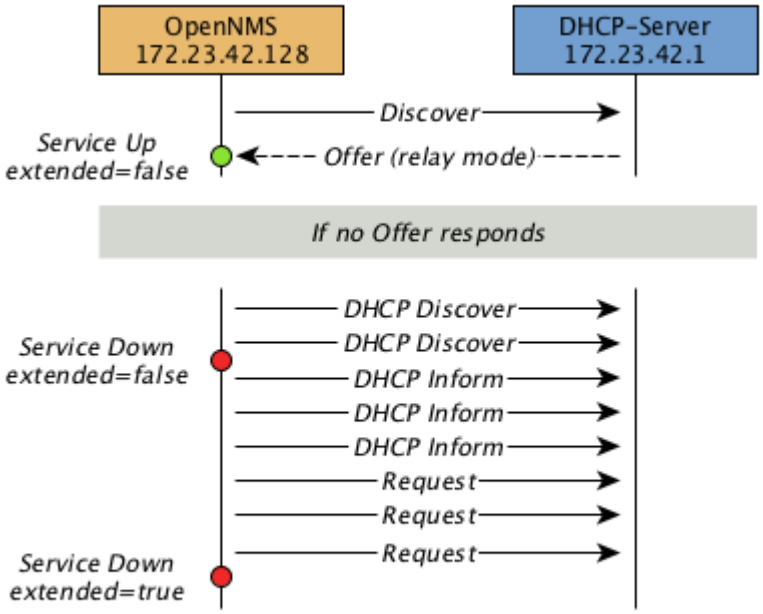


Figure 15. Visualization of DHCP message flow in relay mode

Example testing DHCP server in the same subnet

Example configuration how to configure the monitor in the `poller-configuration.xml`. The monitor will try to send in maximum 3 *DISCOVER* messages and waits 3 seconds for the DHCP server *OFFER* message.

Step 1: Configure a DHCP service in `poller-configuration.xml`

```
<service name="DHCP" interval="300000" user-defined="false" status="on">
 <parameter key="retry" value="2" />
 <parameter key="timeout" value="3000" />
 <parameter key="rrd-repository" value="/opt/opennms/share/rrd/response" />
 <parameter key="rrd-base-name" value="dhcp" />
 <parameter key="ds-name" value="dhcp" />
</service>

<monitor service="DHCP" class-name="org.opennms.protocols.dhcp.monitor.DhcpMonitor"/>
```

Step 2: Enable the OpenNMS Horizon Dhcpd daemon in `service-configuration.xml`

```
<service enabled="true">
  <name>OpenNMS:Name=Dhcpd</name>
  <class-name>org.opennms.netmgt.dhcpd.jmx.Dhcpd</class-name>
  <invoke method="start" pass="1" at="start"/>
  <invoke method="status" pass="0" at="status"/>
  <invoke method="stop" pass="0" at="stop"/>
</service>
```

Step 3: Configure Dhcpd to test a DHCP server in the same subnet as the OpenNMS Horizon server.

```
<DhcpdConfiguration
        port="5818"
        macAddress="00:06:0D:BE:9C:B2"
        myIpAddress="broadcast
        extendedMode="false"
        requestIpAddress="127.0.0.1">
</DhcpdConfiguration>
```

Example testing DHCP server in a different subnet in extended mode

You can use the same monitor in `poller-configuration.xml` as in the example above.

Configure Dhcpd to test DHCP server in a different subnet. The OFFER from the DHCP server is sent to `myIpAddress`.

```
<DhcpdConfiguration
        port="5818"
        macAddress="00:06:0D:BE:9C:B2"
        myIpAddress="10.4.1.234"
        extendedMode="true"
        requestIpAddress="targetSubnet">
</DhcpdConfiguration>
```

If in `extendedMode`, the time required to complete the poll for an unresponsive node is increased by a factor of 3. Thus it is a good idea to limit the number of retries to a small number.

3.6.8. DiskUsageMonitor

The DiskUsageMonitor monitor can be used to test the amount of free space available on certain storages of a node.

The monitor gets information about the available free storage spaces available by inspecting the *hrStorageTable* of the HOST-RESOURCES-MIB.

A storage's description (as found in the corresponding *hrStorageDescr* object) must match the criteria specified by the `disk` and `match-type` parameters to be monitored.

A storage's available free space is calculated using the corresponding *hrStorageSize* and *hrStorageUsed* objects.

The *hrStorageUsed* doesn't account for filesystem reserved blocks (i.e. for the super-user), so DiskUsageMonitor will report the service as unavailable only when the amount of free disk space is actually lower than `free` minus the percentage of reserved filesystem blocks.

This monitor uses *SNMP* to accomplish its work. Therefore systems against which it is to be used must have an SNMP agent supporting the *HOST-RESOURCES-MIB* installed and configured. Most modern *SNMP agents*, including most distributions of the *Net-SNMP agent* and the *SNMP service* that ships with *Microsoft Windows*, support this *MIB*. Out-of-box support for *HOST-RESOURCES-MIB* among commercial *Unix* operating systems may be somewhat spotty.

Monitor facts

Class Name	`org.opennms.netmgt.poller.monitors.DiskUsageMonitor`
Remote Enabled	false, relies on SNMP configuration.

Configuration and Usage

Table 19. Monitor specific parameters for the DiskUsageMonitor

Parameter	Description	Required	Default value
`disk`	A pattern that a storage's description (*hrStorageDescr*) must match to be taken into account.	required	`-`
`free`	The minimum amount of free space that storages matching the criteria must have available. This parameter is evaluated as a percent of the storage's reported maximum capacity.	optional	`15`
`match-type`	The way how the pattern specified by the `disk` parameter must be compared to storages description Must be one of the following symbolic operators: `endswith` : The `disk` parameter's value is evaluated as a string that storages' description must end with; `exact` : The `disk` parameter's value is evaluated as a string that storages" description must exactly match; `regex` : The `disk` parameter's value is evaluated as a regular expression that storages' description must match; `startswith` : The `disk` parameter's value is evaluated as a string that storages' description must start with. Note: Comparisons are case-sensitive	optional	`exact`
`port`	Destination port where the SNMP requests shall be sent.	optional	`from snmp-config.xml`
`retries`	Deprecated. Same as `retry`. Parameter `retry` takes precedence when both are set.	optional	`from snmp-config.xml`
`retry`	Number of polls to attempt.	optional	`from snmp-config.xml`
`timeout`	Timeout in milliseconds for retrieving the values.	optional	`from snmp-config.xml`

Examples

```
<!-- Make sure there's at least 5% of free space available on storages ending with
"/home" -->
<service name="DiskUsage-home" interval="300000" user-defined="false" status="on">
  <parameter key="timeout" value="3000" />
  <parameter key="retry" value="2" />
  <parameter key="disk" value="/home" />
  <parameter key="match-type" value="endsWith" />
  <parameter key="free" value="5" />
</service>
<monitor service="DiskUsage-home" class-name=
"org.opennms.netmgt.poller.monitors.DiskUsageMonitor" />
```

DiskUsageMonitor vs thresholds

Storages' available free space can also be monitored using thresholds if you are already collecting these data.

3.6.9. DnsMonitor

This monitor is build to test the availability of the *DNS service* on remote IP interfaces. The monitor tests the service availability by sending a DNS query for A resource record types against the DNS server to test.

The monitor is marked as *up* if the *DNS Server* is able to send a valid response to the monitor. For multiple records it is possible to test if the number of responses are within a given boundary.

The monitor can be simulated with the command line tool `host`:

```
~ % host -v -t a www.google.com 8.8.8.8
Trying "www.google.com"
Using domain server:
Name: 8.8.8.8
Address: 8.8.8.8#53
Aliases:

;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 9324
;; flags: qr rd ra; QUERY: 1, ANSWER: 5, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;www.google.com.            IN  A

;; ANSWER SECTION:
www.google.com.     283 IN  A   74.125.232.17
www.google.com.     283 IN  A   74.125.232.20
www.google.com.     283 IN  A   74.125.232.19
www.google.com.     283 IN  A   74.125.232.16
www.google.com.     283 IN  A   74.125.232.18

Received 112 bytes from 8.8.8.8#53 in 41 ms
```

TIP: This monitor is intended for testing the availability of a DNS service. If you want to monitor the DNS resolution of some of your nodes from a client's perspective, please use the DNSResolutionMonitor.

Monitor facts

Class Name	`org.opennms.netmgt.poller.monitors.DnsMonitor`
Remote Enabled	true

Configuration and Usage

Table 20. Monitor specific parameters for the DnsMonitor

Parameter	Description	Required	Default value
`retry`	Number of retries before the service is marked as *down*	optional	`0`
`timeout`	Time in milliseconds to wait for the *A Record* response from the server	optional	`5000`
`port`	UDP Port for the DNS server	optional	`53`
`lookup`	DNS *A Record* for lookup test	optional	`localhost`

Parameter	Description	Required	Default value
`fatal-response-codes`	A comma-separated list of numeric DNS response codes that will be considered fatal if present in the server's response. Default value is `2` corresponds to *Server Failed*. A list of codes and their meanings is found in RFC 2929	optional	`2`
`min-answers`	Minmal number of records in the DNS server respone for the given lookup	optional	-
`max-answers`	Maximal number of records in the DNS server respone for the given lookup	optional	-

Examples

The given examples shows how to monitor if the IP interface from a given DNS server resolves a DNS request. This service should be bound to a DNS server which should be able to give a valid DNS respone for DNS request *www.google.com*. The service is *up* if the DNS server gives between `1` and `10` *A record* responses.

Example configuration monitoring DNS request for a given server for www.google.com

```
<service name="DNS-www.google.com" interval="300000" user-defined="false" status="on">
    <parameter key="lookup" value="www.google.com" />
    <parameter key="fatal-response-code" value="2" />
    <parameter key="min-answers" value="1" />
    <parameter key="max-answers" value="10" />
</service>

<monitor service="DNS-www.google.com" class-name=
"org.opennms.netmgt.poller.monitors.DnsMonitor" />
```

3.6.10. DNSResolutionMonitor

The DNS resolution monitor, tests if the node label of an OpenNMS Horizon node can be resolved. This monitor uses the name resolver configuration from the poller configuration or from the operating system where OpenNMS Horizon is running on. It can be used to test a client behavior for a given host name. For example: Create a node with the node label www.google.com and an IP interface. Assigning the DNS resolution monitor on the IP interface will test if www.google.com can be resolved using the DNS configuration defined by the poller. The response from the A record lookup can be any address, it is not verified with the IP address on the OpenNMS Horizon IP interface where the monitor is assigned to.

Monitor facts

Class Name	`org.opennms.netmgt.poller.monitors.DNSResolutionMonitor`
Remote Enabled	true

Configuration and Usage

Table 21. Monitor specific parameters for the DNSResolutionMonitor

Parameter	Description	Required	Default value
`resolution-type`	Type of record for the node label test. Allowed values `v4` for *A records*, `v6` for *AAAA record*, `both` *A* and *AAAA record* must be available, `either` *A* or *AAAA record* must be available.	optional	`either`
`nameserver`	The DNS server to query for the records.	optional	Use the servers defined by the system running OpenNMS Horizon
`retry`	Amount of attempts to resolve the node label before the service goes down	required	-
`timeout`	Time to wait for a *A* and/or *AAAA record* from the system configured *DNS server* before trying a next attempt.	required	-

Examples

The following example shows the possibilities monitoring IPv4 and/or IPv6 for the service configuration:

```
<!-- Assigned service test if the node label is resolved for an A record -->
<service name="DNS-Resolution-v4" interval="300000" user-defined="false" status="on">
    <parameter key="retry" value="2"/>
    <parameter key="timeout" value="2000"/>
    <parameter key="resolution-type" value="v4"/>
    <parameter key="rrd-repository" value="/opt/opennms/share/rrd/response"/>
    <parameter key="rrd-base-name" value="dns-res-v4"/>
    <parameter key="ds-name" value="dns-res-v4"/>
</service>

<!-- Assigned service test if the node label is resolved for an AAAA record using a
specific DNS server -->
<service name="DNS-Resolution-v6" interval="300000" user-defined="false" status="on">
    <parameter key="retry" value="2"/>
    <parameter key="timeout" value="2000"/>
    <parameter key="resolution-type" value="v6"/>
    <parameter key="rrd-repository" value="/opt/opennms/share/rrd/response"/>
    <parameter key="rrd-base-name" value="dns-res-v6"/>
    <parameter key="ds-name" value="dns-res-v6"/>
    <parameter key="nameserver" value="8.8.8.8"/>
</service>

<!-- Assigned service test if the node label is resolved for an AAAA record AND A
```

```
record -->
<service name="DNS-Resolution-v4-and-v6" interval="300000" user-defined="false"
status="on">
    <parameter key="retry" value="2"/>
    <parameter key="timeout" value="2000"/>
    <parameter key="resolution-type" value="both"/>
    <parameter key="rrd-repository" value="/opt/opennms/share/rrd/response"/>
    <parameter key="rrd-base-name" value="dns-res-both"/>
    <parameter key="ds-name" value="dns-res-both"/>
</service>

<!-- Assigned service test if the node label is resolved for an AAAA record OR A
record -->
<service name="DNS-Resolution-v4-or-v6" interval="300000" user-defined="false" status
="on">
    <parameter key="retry" value="2"/>
    <parameter key="timeout" value="2000"/>
    <parameter key="resolution-type" value="either"/>
    <parameter key="rrd-repository" value="/opt/opennms/share/rrd/response"/>
    <parameter key="rrd-base-name" value="dns-res-either"/>
    <parameter key="ds-name" value="dns-res-either"/>
</service>

<monitor service="DNS-Resolution-v4" class-name=
"org.opennms.netmgt.poller.monitors.DNSResolutionMonitor" />
<monitor service="DNS-Resolution-v6" class-name=
"org.opennms.netmgt.poller.monitors.DNSResolutionMonitor" />
<monitor service="DNS-Resolution-v4-and-v6" class-name=
"org.opennms.netmgt.poller.monitors.DNSResolutionMonitor" />
<monitor service="DNS-Resolution-v4-or-v6" class-name=
"org.opennms.netmgt.poller.monitors.DNSResolutionMonitor" />
```

To have response time graphs for the name resolution you have to configure RRD graphs for the given ds-names (`dns-res-v4`, `dns-res-v6`, `dns-res-both`, `dns-res-either`) in '$OPENNMS_HOME/etc/response-graph.properties'.

DNSResolutionMonitor vs DnsMonitor

The DNSResolutionMonitor is used to measure the availability and record outages of a name resolution from client perspective. The service is mainly used for websites or similar public available resources. It can be used in combination with the Page Sequence Monitor to give a hint if a website isn't available for DNS reasons.

The DnsMonitor on the other hand is a test against a specific DNS server. In OpenNMS Horizon the DNS server is the node and the DnsMonitor will send a lookup request for a given A record to the DNS server IP address. The service goes down if the DNS server doesn't have a valid A record in his zone database or as some other issues resolving A records.

3.6.11. FtpMonitor

The FtpMonitor is able to validate ftp connection dial-up processes. The monitor can test ftp server on multiple ports and specific login data.

The service using the FtpMonitor is *up* if the FTP server responds with return codes between 200 and 299. For special cases the service is also marked as *up* for 425 and 530.

Monitor facts

Class Name	`org.opennms.netmgt.poller.monitors.FtpMonitor`
Remote Enabled	`true`

Configuration and Usage

Table 22. Monitor specific parameters for the FtpMonitor.

Parameter	Description	Required	Default value
`retry`	Number of attempts to get a valid FTP response/response-text	optional	`0`
`timeout`	Timeout in milliseconds for TCP connection establishment.	optional	`3000`
`port`	A list of TCP ports to which connection shall be tried.	optional	`20,21`
`password`	This parameter is meant to be used together with the `user` parameter to perform basic authentication. This parameter specify to password to be used. The `user` and `password` parameters are ignored when the `basic-authentication` parameter is defined.	optional	`empty string`
`userid`	This parameter is meant to be used together with the `password` parameter to perform basic authentication. This parameter specify to user ID to be used. The `userid` and `password` parameters are ignored when the `basic-authentication` parameter is defined.	optional	`-`

Examples

Some example configuration how to configure the monitor in the 'poller-configuration.xml'

```
<service name="FTP" interval="300000" user-defined="false" status="on">
 <parameter key="retry" value="1"/>
 <parameter key="timeout" value="3000"/>
 <parameter key="port" value="21"/>
 <parameter key="userid" value=""/>
 <parameter key="password" value=""/>
</service>

<service name="FTP-Customer" interval="300000" user-defined="false" status="on">
 <parameter key="retry" value="1"/>
 <parameter key="timeout" value="3000"/>
 <parameter key="port" value="21"/>
 <parameter key="userid" value="Customer"/>
 <parameter key="password" value="MySecretPassword"/>
</service>

<monitor service="FTP" class-name="org.opennms.netmgt.poller.monitors.FtpMonitor"/>
<monitor service="FTP-Customer" class-name=
"org.opennms.netmgt.poller.monitors.FtpMonitor"/>
```

Hint

Comment from FtpMonitor source

Also want to accept the following ERROR message generated by some FTP servers following a QUIT command without a previous successful login: "530 QUIT : User not logged in. Please login with USER and PASS first."

Also want to accept the following ERROR message generated by some FTP servers following a QUIT command without a previously successful login: "425 Session is disconnected."

See also: http://tools.ietf.org/html/rfc959

3.6.12. HostResourceSwRunMonitor

This monitor test the running state of one or more processes. It does this via SNMP by inspecting the *hrSwRunTable* of the HOST-RESOURCES-MIB. The test is done by matching a given process as *hrSwRunName* against the numeric value of the *hrSwRunState*.

This monitor uses *SNMP* to accomplish its work. Therefore systems against which it is to be used must have an SNMP agent installed and configured. Furthermore, the *SNMP agent* on the system must support the *HOST-RESOURCES-MIB*. Most modern *SNMP agents*, including most distributions of the *Net-SNMP agent* and the *SNMP service* that ships with *Microsoft Windows*, support this *MIB*. Out-of-box support for *HOST-RESOURCES-MIB* among commercial *Unix* operating systems may be somewhat spotty.

Monitor facts

Class Name	`org.opennms.netmgt.poller.monitors.HostResourceSwRunMonitor`

Remote Enabled	false

Configuration and Usage

Table 23. Monitor specific parameters for the HostResourceSwRunMonitor

Parameter	Description	Required	Default value
`port`	The port of the *SNMP agent* of the server to test.	optional	`from snmp-config.xml`
`retry`	Number of attempts to get a valid response before marking the service as *down*.	optional	`from snmp-config.xml`
`timeout`	Timeout in milliseonds wating for the *SNMP response* for the process run state from the agent.	optional	`from snmp-config.xml`
`service-name`	The name of the process to be monitored. This parameter's value is case-sensitive and is evaluated as an exact match.	required	-
`match-all`	If the process name appears multiple times in the *hrSwRunTable*, and this parameter is set to `true`, then all instances of the named process must match the value specified for `run-level`.	optional	`false`
`run-level`	The maximum allowable value of *hrSWRunStatus* among *running(1)*, *runnable(2)* = waiting for resource *notRunnable(3)* = loaded but waiting for event *invalid(4)* = not loaded	optional	`2`
`service-name-oid`	The numeric object identifier (OID) from which process names are queried. Defaults to *hrSwRunName* and should never be changed under normal circumstances. That said, changing it to *hrSwRunParameters* (`.1.3.6.1.2.1.25.4.2.1.5`) is often helpful when dealing with processes running under *Java Virtual Machines* which all have the same process name *java*.	optional	`.1.3.6.1.2.1.25.4.2.1.2`
`service-status-oid`	The numeric object identifier (OID) from which run status is queried. Defaults to *hrSwRunStatus* and should never be changed under normal circumstances.	optional	`.1.3.6.1.2.1.25.4.2.1.7`

Examples

The following example shows how to monitor the process called *httpd* running on a server using this monitor. The configuration in `poller-configuration.xml` has to be defined as the following:

```
<service name="Process-httpd" interval="300000" user-defined="false" status="on">
    <parameter key="retry" value="3"/>
    <parameter key="timeout" value="3000"/>
    <parameter key="service-name" value="httpd"/> ①
    <parameter key="run-level" value="3"/> ②
    <parameter key="match-all" value="true"/> ③
</service>

<monitor service="Process-httpd" class-name=
"org.opennms.netmgt.poller.monitors.HostResourceSwRunMonitor"/>
```

① Name of the process on the system

② Test the state if the process is in a valid state, i.e. have a `run-level` no higher than *notRunnable(3)*

③ If the *httpd* process runs multiple times the test is done for each instance of the process.

3.6.13. HttpMonitor

The HTTP monitor tests the response of an HTTP server on a specific HTTP 'GET' command. During the poll, an attempt is made to connect on the specified port(s). The monitor can test web server on multiple ports. By default the a test is made against port 80, 8080 and 8888. If the connection request is successful, an HTTP 'GET' command is sent to the interface. The response is parsed and a return code extracted and verified.

Monitor facts

Class Name	`org.opennms.netmgt.poller.monitors.HttpMonitor`
Remote Enabled	true

Configuration and Usage

Table 24. Monitor specific parameters for the HttpMonitor

Parameter	Description	Required	Default value
`basic-authentication`	Authentication credentials to perform basic authentication. Credentials should comply to RFC1945 section 11.1, without the Base64 encoding part. That's: be a string made of the concatenation of: 1- the user ID; 2- a colon; 3- the password. `basic-authentication` takes precedence over the `user` and `password` parameters.	optional	-

Parameter	Description	Required	Default value
`header[0-9]+`	Additional headers to be sent along with the request. Example of valid parameter's names are `header0`, `header1` and `header180`. `header` is **not** a valid parameter name.	optional	`-`
`host-name`	Specify the *Host* header's value.	optional	`-`
`nodelabel-host-name`	If the `host-name` parameter isn't set and the `resolve-ip` parameter is set to `false`, then OpenNMS Horizon will use the node's label to set the *Host* header's value if this parameter is set to `true`. Otherwise, OpenNMS Horizon will fall back using the node interface's IP address as *Host* header value.	optional	`false`
`password`	This parameter is meant to be used together with the `user` parameter to perform basic authentication. This parameter specify to password to be used. The `user` and `password` parameters are ignored when the `basic-authentication` parameter is defined.	optional	`empty string`
`port`	A list of TCP ports to which connection shall be tried.	optional	`80,8080,8888`
`retry`	Number of attempts to get a valid HTTP response/response-text	optional	`0`
`resolve-ip`	If the `host-name` parameter isn't set and this parameter is set to `true`, OpenNMS Horizon will use DNS to resolve the node interface's IP address, and use the result to set the *Host* header's value. When set to `false` and the `host-name` parameter isn't set, OpenNMS Horizon will try to use the `nodelabel-host-name` parameter to set the *Host* header's value.	optional	`false`
`response`	A comma-separated list of acceptable HTTP response code ranges. Example: `200-202,299`	optional	If the `url` parameter is set to `/`, the default value for this parameter is `100-499`, otherwise it's `100-399`.

Parameter	Description	Required	Default value
`response-text`	Text to look for in the response body. This will be matched against every line, and it will be considered a success at the first match. If there is a `~` at the beginning of the parameter, the rest of the string will be used as a regular expression pattern match, otherwise the match will be a substring match. The regular expression match is anchored at the beginning and end of the line, so you will likely need to put a `.*` on both sides of your pattern unless you are going to be matching on the entire line.	optional	`-`
`timeout`	Timeout in milliseconds for TCP connection establishment.	optional	`3000`
`url`	URL to be retrieved via the HTTP 'GET' command	optional	`/`
`user`	This parameter is meant to be used together with the `password` parameter to perform basic authentication. This parameter specify to user ID to be used. The `user` and `password` parameters are ignored when the `basic-authentication` parameter is defined.	optional	`-`
`user-agent`	Allows you to set the *User-Agent* HTTP header (see also RFC2616 section 14.43).	optional	`OpenNMS HttpMonitor`
`verbose`	When set to *true*, full communication between client and the webserver will be logged (with a log level of `DEBUG`).	optional	`-`

Examples

```
<!-- Test HTTP service on port 80 only -->
<service name="HTTP" interval="300000" user-defined="false" status="on">
  <parameter key="retry" value="2"/>
  <parameter key="timeout" value="3000"/>
  <parameter key="port" value="80"/>
  <parameter key="url" value="/"/>
</service>

<!-- Test for virtual host opennms.com running -->
<service name="OpenNMSdotCom" interval="300000" user-defined="false" status="on">
  <parameter key="retry" value="1"/>
  <parameter key="timeout" value="3000"/>
  <parameter key="port" value="80"/>
  <parameter key="host-name" value="opennms.com"/>
  <parameter key="url" value="/solutions"/>
  <parameter key="response" value="200-202,299"/>
  <parameter key="response-text" value="~.*[Cc]onsulting.*"/>
</service>

<!-- Test for instance of OpenNMS 1.2.9 running -->
<service name="OpenNMS-129" interval="300000" user-defined="false" status="on">
  <parameter key="retry" value="1"/>
  <parameter key="timeout" value="3000"/>
  <parameter key="port" value="8080"/>
  <parameter key="url" value="/opennms/event/list"/>
  <parameter key="basic-authentication" value="admin:admin"/>
  <parameter key="response" value="200"/>
</service>

<monitor service="HTTP" class-name="org.opennms.netmgt.poller.monitors.HttpMonitor" />
<monitor service="OpenNMSdotCom" class-name=
"org.opennms.netmgt.poller.monitors.HttpMonitor" />
<monitor service="OpenNMS-129" class-name=
"org.opennms.netmgt.poller.monitors.HttpMonitor" />
```

Testing filtering proxies with HttpMonitor

If you have a filtering proxy server that is supposed to allow retrieval of some URLs but deny others, you can use the HttpMonitor to verify this behavior.

Let's say that our proxy server is running on TCP port 3128, and that we should always be able to retrieve http://www.opennms.org/ but never http://www.myspace.com/ (hey, this is a workplace after all!). To test this behaviour, one could create the following service monitors:

```
<service name="HTTP-Allow-opennms.org" interval="300000" user-defined="false" status=
"on">
  <parameter key="retry" value="1"/>
  <parameter key="timeout" value="3000"/>
  <parameter key="port" value="3128"/>
  <parameter key="url" value="http://www.opennms.org/"/>
  <parameter key="response" value="200-399"/>
</service>

<service name="HTTP-Block-myspace.com" interval="300000" user-defined="false" status=
"on">
  <parameter key="retry" value="1"/>
  <parameter key="timeout" value="3000"/>
  <parameter key="port" value="3128"/>
  <parameter key="url" value="http://www.myspace.com/"/>
  <parameter key="response" value="400-599"/>
</service>

<monitor service="HTTP-Allow-opennms.org" class-name=
"org.opennms.netmgt.poller.monitors.HttpMonitor"/>
<monitor service="HTTP-Block-myspace.com" class-name=
"org.opennms.netmgt.poller.monitors.HttpMonitor"/>
```

3.6.14. HttpPostMonitor

If it is required to *HTTP POST* any arbitrary content to a remote *URI*, the HttpPostMonitor can be used. A use case is to HTTP POST to a SOAP endpoint.

Monitor facts

Class Name	`org.opennms.netmgt.poller.monitors.HttpPostMonitor`
Remote Enabled	`false`

Configuration and Usage

Table 25. Monitor specific parameters for the HttpPostMonitor

Parameter	Description	Required	Default value
`payload`	The body of the POST, for example properly escaped XML.	required	-
`auth-password`	The password to use for HTTP BASIC auth.	optional	-
`auth-username`	The username to use for HTTP BASIC auth.	optional	-

Parameter	Description	Required	Default value
banner	A string that is matched against the response of the HTTP POST. If the output contains the banner, the service is determined as up. Specify a regex by starting with ~.	optional	-
charset	Set the character set for the POST.	optional	UTF-8
mimetype	Set the mimetype for the POST.	optional	text/xml
port	The port for the web server where the POST is send to.	optional	80
scheme	The connection scheme to use.	optional	http
usesslfilter	Enables or disables the SSL ceritificate validation. true - false	optional	false
uri	The uri to use during the POST.	optional	/

Examples

The following example would create a POST that contains the payload *Word*.

```
<service name="MyServlet" interval="300000" user-defined="false" status="on">
  <parameter key="banner" value="Hello"/>
  <parameter key="port" value="8080"/>
  <parameter key="uri" value="/MyServlet">
  <parameter key="payload" value="World"/>
  <parameter key="retry" value="1"/>
  <parameter key="timeout" value="30000"/>
</service>
<monitor service="MyServlet" class-name=
"org.opennms.netmgt.poller.monitors.HttpPostMonitor"/>
```

The resulting POST looks like this:

```
POST /MyServlet HTTP/1.1
Content-Type: text/xml; charset=utf-8
Host: <ip_addr_of_interface>:8080
Connection: Keep-Alive

World
```

3.6.15. HttpsMonitor

The HTTPS monitor tests the response of an SSL-enabled HTTP server. The HTTPS monitor is an SSL-enabled extension of the HTTP monitor with a default TCP port value of 443. All HttpMonitor parameters apply, so please refer to HttpMonitor's documentation for more information.

Monitor facts

Class Name	`org.opennms.netmgt.poller.monitors.HttpsMonitor`
Remote Enabled	true

Configuration and Usage

Table 26. Monitor specific parameters for the HttpsMonitor

Parameter	Description	Required	Default value
`port`	A list of TCP ports to which connection shall be tried.	optional	`443`

Examples

```
<!-- Test HTTPS service on port 8443 -->
<service name="HTTPS" interval="300000" user-defined="false" status="on">
  <parameter key="retry" value="2"/>
  <parameter key="timeout" value="3000"/>
  <parameter key="port" value="8443"/>
  <parameter key="url" value="/"/>
</service>

<monitor service="HTTPS" class-name="org.opennms.netmgt.poller.monitors.HttpsMonitor"
/>
```

3.6.16. IcmpMonitor

The ICMP monitor tests for ICMP service availability by sending *echo request* ICMP messages. The service is considered available when the node sends back an *echo reply* ICMP message within the specified amount of time.

Monitor facts

Class Name	`org.opennms.netmgt.poller.monitors.IcmpMonitor`
Remote Enabled	true with some restrictions (see below)

Configuration and Usage

Table 27. Monitor specific parameters for the IcmpMonitor

Parameter	Description	Required	Default value
packet-size	Number of bytes of the ICMP packet to send.	optional	64
retry	Number of attempts to get a response.	optional	2
timeout	Time in milliseconds to wait for a response.	optional	800
thresholding-enabled	Enables ICMP thresholding	optional	true

Examples

```
<service name="ICMP" interval="300000" user-defined="false" status="on">
  <parameter key="retry" value="2"/>
  <parameter key="timeout" value="3000"/>
  <parameter key="rrd-repository" value="/var/lib/opennms/rrd/response"/>
  <parameter key="rrd-base-name" value="icmp"/>
  <parameter key="ds-name" value="icmp"/>
</service>
<monitor service="ICMP" class-name="org.opennms.netmgt.poller.monitors.IcmpMonitor"/>
```

Note on Remote Poller

The IcmpMonitor needs the JNA ICMP implementation to function on remote poller. Though, corner cases exist where the IcmpMonitor monitor won't work on remote poller. Examples of such corner cases are: Windows when the remote poller isn't running has administrator, and Linux on ARM / Rasperry Pi. JNA is the default ICMP implementation used in the remote poller.

3.6.17. ImapMonitor

This monitor checks if an IMAP server is functional. The test is done by initializing a very simple IMAP conversation. The ImapMonitor establishes a TCP connection, sends a logout command and test the IMAP server responses.

The behavior can be simulated with `telnet`:

```
telnet mail.myserver.de 143
Trying 62.108.41.197...
Connected to mail.myserver.de.
Escape character is '^]'.
* OK [CAPABILITY IMAP4rev1 LITERAL+ SASL-IR LOGIN-REFERRALS ID ENABLE IDLE STARTTLS
LOGINDISABLED] Dovecot ready. ①
ONMSPOLLER LOGOUT ②
* BYE Logging out ③
ONMSPOLLER OK Logout completed.
Connection closed by foreign host.
```

① Test IMAP server banner, it has to start `* OK` to be *up*

② Sending a `ONMSPOLLER LOGOUT`

③ Test server responds with, it has to start with `* BYE` to be *up*

If one of the tests in the sample above fails the service is marked *down*.

Monitor facts

Class Name	`org.opennms.netmgt.poller.monitors.ImapMonitor`
Remote Enabled	false

Configuration and Usage

Table 28. Monitor specific parameters for the ImapMonitor

Parameter	Description	Required	Default value
`retry`	Number of attempts to get a valid IMAP response	optional	`0`
`timeout`	Time in milliseconds to wait retrieving the banner from TCP connection before trying a next attempt.	optional	`3000`
`port`	The port of the IMAP server.	optional	`143`

Examples

Some example configuration how to configure the monitor in the `poller-configuration.xml`

```
<!-- Test IMAP service on port 143 only -->
<service name="IMAP" interval="300000" user-defined="false" status="on">
  <parameter key="retry" value="1"/>
  <parameter key="port" value="143"/>
  <parameter key="timeout" value="3000"/>
</service>

<monitor service="IMAP" class-name="org.opennms.netmgt.poller.monitors.ImapMonitor" />
```

3.6.18. JCifsMonitor

This monitor allows to test a file sharing service based on the CIFS/SMB protocol.

This monitor is not installed by default. You have to install `opennmms-plugin-protocol-cifs` from your OpenNMS Horizon installation repository.

With the *JCIFS* monitor you have different possibilities to test the availability of the *JCIFS* service:

With the *JCifsMonitor* it is possible to run tests for the following use cases:

- share is available in the network
- a given file exists in the share
- a given folder exists in the share
- a given folder should contain at least one (1) file
- a given folder folder should contain no (0) files
- by testing on files and folders, you can use a regular expression to ignore specific file and folder names from the test

A network resource in SMB like a file or folder is addressed as a UNC Path.

```
\\server\share\folder\file.txt
```

The Java implementation *jCIFS*, which implements the *CIFS/SMB* network protocol, uses *SMB* URLs to access the network resource. The same resource as in our example would look like this as an SMB URL:

```
smb://workgroup;user:password@server/share/folder/file.txt
```

The *JCifsMonitor* can **not** test:

- file contains specific content
- a specific number of files in a folder, for example folder should contain exactly / more or less than x files
- Age or modification time stamps of files or folders
- Permissions or other attributes of files or folders

Monitor facts

Class Name	`org.opennms.netmgt.poller.monitors.JCifsMonitor`
Remote Enabled	false

Configuration and Usage

Table 29. Monitor specific parameters for the JCifsMonitor

Parameter	Description	Required	Default value
`retry`	Number of retries before the service is marked as *down*.	optional	`0`
`timeout`	Time in milliseconds to wait for the SMB service.	optional	`3000`

Parameter	Description	Required	Default value
`domain`	Windows domain where the user is located. You don't have to use the domain parameter if you use local user accounts.	optional	empty String
`username`	Username to access the resource over a network	optional	empty String
`password`	Password for the user	optional	empty String
`path`	Path to the resource you want to test	required	empty String
`mode`	The test mode which has the following options `path_exist`: Service is *up* if the resource is accessible `path_not_exist`: Service is *up* if the resource is **not** accessible `folder_empty`: Service is *up* if the folder is empty (0 files) `folder_not_empty`: Service is *up* if the folder has at least one file	optional	`path_exist`
`smbHost`	Override the IP address of the SMB url to check shares on different file servers.	optional	empty String
`folderIgnoreFiles`	Ignore specific files in folder with regular expression. This parameter will just be applied on `folder_empty` and `folder_not_empty`, otherwise it will be ignored.	optional	-

It makes little sense to have retries higher than `1`. It is a waste of resources during the monitoring.

Please consider, if you are accessing shares with Mac OSX you have some side effects with the hidden file '.DS_Store.' It could give you false positives in monitoring, you can use then the `folderIgnoreFiles` parameter.

Example test existence of a file

This example shows how to configure the *JCifsMonitor* to test if a file share is available over a network. For this example we have access to a share for error logs and we want to get an outage if we have any error log files in our folder. The share is named *log*. The service should go back to normal if the error log file is deleted and the folder is empty.

JCifsMonitor configuration to test that a shared folder is empty

```
<service name="CIFS-ErrorLog" interval="30000" user-defined="true" status="on">
    <parameter key="retry" value="1" />
    <parameter key="timeout" value="3000" />
    <parameter key="domain" value="contoso" /> ①
    <parameter key="username" value="MonitoringUser" /> ②
    <parameter key="password" value="MonitoringPassword" /> ③
    <parameter key="path" value="/fileshare/log/" /> ④
    <parameter key="mode" value="folder_empty" /> ⑤
</service>

<monitor service="CIFS-ErrorLog" class-name=
"org.opennms.netmgt.poller.monitors.JCifsMonitor" />
```

① Name of the SMB or Microsoft Windows Domain

② User for accessing the share

③ Password for accessing the share

④ Path to the folder inside of the share as part of the SMB URL

⑤ Mode is set to `folder_empty`

3.6.19. JDBCMonitor

The *JDBCMonitor* checks that it is able to connect to a database and checks if it is able to get the database catalog from that database management system (DBMS). It is based on the JDBC technology to connect and communicate with the database.

Monitor facts

Class Name	`org.opennms.netmgt.poller.monitors.JDBCMonitor`
Remote Enabled	true

Configuration and Usage

Table 30. Monitor specific parameters for the JDBCMonitor

Parameter	Description	Required	Default value
`driver`	JDBC driver class to use	required	`com.sybase.jdbc2.jdbc.SybDriver`
`url`	JDBC Url to connect to.	required	`jdbc:sybase:Tds:OPENNMS_JDBC_HOSTNAME/tempdb`
`user`	Database user	required	`sa`

Parameter	Description	Required	Default value
`password`	Database password	required	`empty string`
`timeout`	Timeout in ms for the database connection	optional	`3000`
`retries`	How many retries should be performed before failing the test	optional	`0`

The *OPENNMS_JDBC_HOSTNAME* is replaced in the *url* parameter with the IP or resolved hostname of the interface the monitored service is assigned to.

Provide the database driver

The *JDBCMonitor* is based on *JDBC* and requires a JDBC driver to communicate with any database. Due to the fact that OpenNMS Horizon itself uses a PostgreSQL database, the PostgreSQL JDBC driver is available out of the box. For all other database systems a compatible JDBC driver has to be provided to OpenNMS Horizon as a *jar-file*. To provide a JDBC driver place the *driver-jar* in the `opennms/lib` folder of your OpenNMS Horizon. To use the *JDBCMonitor* from a remote poller, the *driver-jar* has to be provided to the *Remote Poller* too. This may be tricky or impossible when using the *Java Webstart Remote Poller*, because of code signing requirements.

Examples

The following example checks if the PostgreSQL database used by OpenNMS Horizon is available.

```
<service name="OpenNMS-DBMS" interval="30000" user-defined="true" status="on">
  <parameter key="driver" value="org.postgresql.Driver"/>
  <parameter key="url" value="jdbc:postgresql://OPENNMS_JDBC_HOSTNAME:5432/opennms"/>
  <parameter key="user" value="opennms"/>
  <parameter key="password" value="opennms"/>
</service>

<monitor service="OpenNMS-DBMS" class-name=
"org.opennms.netmgt.poller.monitors.JDBCMonitor" />
```

3.6.20. JDBCStoredProcedureMonitor

The *JDBCStoredProcedureMonitor* checks the result of a stored procedure in a remote database. The result of the stored procedure has to be a boolean value (representing true or false). The service associated with this monitor is marked as up if the stored procedure returns true and it is marked as down in all other cases. It is based on the JDBC technology to connect and communicate with the database.

Monitor facts

Class Name	`org.opennms.netmgt.poller.monitors.JDBCStoredProcedureMonitor`
Remote Enabled	false

Configuration and Usage

Table 31. Monitor specific parameters for the JDBCStoredProcedureMonitor

Parameter	Description	Required	Default value
`driver`	JDBC driver class to use	required	`com.sybase.jdbc2.jdbc.SybDriver`
`url`	JDBC Url to connect to.	required	`jdbc:sybase:Tds:OPENNMS_JDBC_HOSTNAME/tempdb`
`user`	Database user	required	`sa`
`password`	Database password	required	`empty string`
`timeout`	Timeout in ms for the database connection	optional	`3000`
`retries`	How many retries should be performed before failing the test	optional	`0`
`stored-procedure`	Name of the database stored procedure to call	required	`-`
`schema`	Name of the database schema in which the stored procedure is	optional	`test`

The *OPENNMS_JDBC_HOSTNAME* is replaced in the *url* parameter with the IP or resolved hostname of the interface the monitored service is assigned to.

Provide the database driver

The *JDBCStoredProcedureMonitor* is based on *JDBC* and requires a *JDBC driver* to communicate with any database. Due to the fact that OpenNMS Horizon itself uses a *PostgreSQL* database, the *PostgreSQL JDBC driver* is available out of the box. For all other database systems a compatible *JDBC driver* has to be provided to OpenNMS Horizon as a *jar-file*. To provide a *JDBC driver* place the *driver-jar* in the `opennms/lib` folder of your OpenNMS Horizon. To use the *JDBCStoredProcedureMonitor* from a remote poller, the *driver-jar* has to be provided to the *Remote Poller* too. This may be tricky or impossible when using the *Java Webstart Remote Poller*, because of code signing requirements.

Examples

The following example checks a stored procedure added to the *PostgreSQL* database used by

OpenNMS Horizon. The stored procedure returns true as long as less than 250000 events are in the events table of OpenNMS Horizon.

Stored procedure which is used in the monitor

```
CREATE OR REPLACE FUNCTION eventlimit_sp() RETURNS boolean AS
$BODY$DECLARE
num_events integer;
BEGIN
    SELECT COUNT(*) into num_events from events;
    RETURN num_events > 250000;
END;$BODY$
LANGUAGE plpgsql VOLATILE NOT LEAKPROOF
COST 100;
```

```
<service name="OpenNMS-DB-SP-Event-Limit" interval="300000" user-defined="true"
status="on">
  <parameter key="driver" value="org.postgresql.Driver"/>
  <parameter key="url" value="jdbc:postgresql://OPENNMS_JDBC_HOSTNAME:5432/opennms"/>
  <parameter key="user" value="opennms"/>
  <parameter key="password" value="opennms"/>
  <parameter key="stored-procedure" value="eventlimit_sp"/>
  <parameter key="schema" value="public"/>
</service>

<monitor service="OpenNMS-DB-SP-Event-Limit" class-name=
"org.opennms.netmgt.poller.monitors.JDBCStoredProcedureMonitor"/>
```

3.6.21. JDBCQueryMonitor

The *JDBCQueryMonitor* runs an SQL query against a database and is able to verify the result of the query. A read-only connection is used to run the SQL query, so the data in the database is not altered. It is based on the JDBC technology to connect and communicate with the database.

Monitor facts

Class Name	`org.opennms.netmgt.poller.monitors.JDBCQueryMonitor`
Remote Enabled	false

Configuration and Usage

Table 32. Monitor specific parameters for the JDBCQueryMonitor

Parameter	Description	Required	Default value
`driver`	JDBC driver class to use	required	`com.sybase.jdbc2.jdbc.SybDriver`

Parameter	Description	Required	Default value
`url`	JDBC URL to connect to.	required	`jdbc:sybase:Tds:OPENNMS_JDBC_HOSTNAME/tempdb`
`user`	Database user	required	`sa`
`password`	Database password	required	`empty string`
`query`	The SQL query to run	required	`-`
`action`	What evaluation action to perform	required	`row_count`
`column`	The result column to evaluate against	optional	`-`
`operator`	Operator to use for the evaluation	required	`>=`
`operand`	The operand to compare against the SQL query result	required	depends on the action
`message`	The message to use if the service is down. Both operands and the operator are added to the message too.	optional	generic message depending on the action
`timeout`	Timeout in ms for the database connection	optional	`3000`
`retries`	How many retries should be performed before failing the test	optional	`0`

The OPENNMS_JDBC_HOSTNAME is replaced in the url parameter with the IP or resolved hostname of the interface the monitored service is assigned to.

Table 33. Available action parameters and their default operand

Parameter	Description	Default operand
`row_count`	The number of returned rows is compared, not a value of the resulting rows	`1`
`compare_string`	Strings are always checked for equality with the operand	`-`
`compare_int`	An integer from a column of the first result row is compared	`1`

Table 34. Available operand parameters

Parameter	XML entity to use in XML configs
=	=
<	<
>	>
!=	!=
⇐	<=
>=	>=

Evaluating the action - operator - operand

Only the first result row returned by the SQL query is evaluated. The evaluation can be against the value of one column or the number of rows returned by the SQL query.

Provide the database driver

The *JDBCQueryMonitor* is based on *JDBC* and requires a JDBC driver to communicate with any database. Due to the fact that OpenNMS Horizon itself uses a PostgreSQL database, the PostgreSQL JDBC driver is available out of the box. For all other database systems a compatible JDBC driver has to be provided to OpenNMS Horizon as a *jar-file*. To provide a JDBC driver place the *driver-jar* in the `opennms/lib` folder of your OpenNMS Horizon. To use the *JDBCQueryMonitor* from a remote poller, the *driver-jar* has to be provided to the *Remote Poller* too. This may be tricky or impossible when using the *Java Webstart Remote Poller*, because of code signing requirements.

Examples

The following example checks if the number of events in the OpenNMS Horizon database is fewer than 250000.

```
<service name="OpenNMS-DB-Event-Limit" interval="30000" user-defined="true" status="
on">
  <parameter key="driver" value="org.postgresql.Driver"/>
  <parameter key="url" value="jdbc:postgresql://OPENNMS_JDBC_HOSTNAME:5432/opennms"/>
  <parameter key="user" value="opennms"/>
  <parameter key="password" value="opennms"/>
  <parameter key="query" value="select eventid from events" />
  <parameter key="action" value="row_count" />
  <parameter key="operand" value="250000" />
  <parameter key="operator" value="&lt;" />
  <parameter key="message" value="too many events in OpenNMS database" />
</service>

<monitor service="OpenNMS-DB-Event-Limit" class-name=
"org.opennms.netmgt.poller.monitors.JDBCQueryMonitor" />
```

3.6.22. JolokiaBeanMonitor

The JolokiaBeanMonitor is a JMX monitor specialized for the use with the Jolokia framework. If it is required to execute a method via *JMX* or poll an attribute via *JMX*, the *JolokiaBeanMonitor* can be used. It requires a fully installed and configured *Jolokia agent* to be deployed in the JVM container. If required it allows attribute names, paths, and method parameters to be provided additional arguments to the call. To determine the status of the service the *JolokiaBeanMonitor* relies on the output to be matched against a banner. If the banner is part of the output the status is interpreted as up. If the banner is not available in the output the status is determined as *down*. Banner matching supports regular expression and substring match.

Monitor facts

Class Name	`org.opennms.netmgt.poller.monitors.JolokiaBeanMonitor`
Remote Enabled	false

Configuration and Usage

Table 35. Monitor specific parameters for the JolokiaBeanMonitor

Parameter	Description	Required	Default value
`beanname`	The bean name to query against.	required	`-`
`attrname`	The name of the JMX attribute to scrape.	optional (`attrname` or `methodname` must be set)	`-`
`attrpath`	The attribute path.	optional	`-`
`auth-username`	The username to use for HTTP BASIC auth.	optional	`-`
`auth-password`	The password to use for HTTP BASIC auth.	optional	`-`
`banner`	A string that is match against the output of the system-call. If the output contains the banner, the service is determined as *up*. Specify a regex by starting with `~`.	optional	`-`
`input1`	Method input	optional	`-`
`input2`	Method input	optional	`-`
`methodname`	The name of the bean method to execute, output will be compared to banner.	optional (`attrname` or `methodname` must be set)	`-`
`port`	The port of the jolokia agent.	optional	`8080`
`url`	The jolokia agent url. Defaults to "http://<ipaddr>:<port>/jolokia"	optional	`-`

Table 36. Variables which can be used in the configuration

Variable	Description
`${ipaddr}`	IP-address of the interface the service is bound to.
`${port}`	Port the service it bound to.

Examples

Some example configuration how to configure the monitor in the `poller-configuration.xml`

```
<parameter key="url" value="http://${ipaddr}:${port}/jolokia"/>
<parameter key="url" value="https://${ipaddr}:${port}/jolokia"/>
```

AttrName vs MethodName

The JolokiaBeanMonitor has two modes of operation. It can either scrape an attribute from a bean, or execute a method and compare output to a banner. The method execute is useful when your application has it's own test methods that you would like to trigger via OpenNMS Horizon.

The args to execute a test method called "superTest" that take in a string as input would look like this:

```
<parameter key="beanname" value="MyBean" />
<parameter key="methodname" value="superTest" />
<parameter key="input1" value="someString"/>
```

The args to scrape an attribute from the same bean would look like this:

```
<parameter key="beanname" value="MyBean" />
<parameter key="attrname" value="upTime" />
```

3.6.23. LdapMonitor

The LDAP monitor tests for LDAP service availability. The LDAP monitor first tries to establish a TCP connection on the specified port. Then, if it succeeds, it will attempt to establish an LDAP connection and do a simple search. If the search returns a result within the specified timeout and attempts, the service will be considered available. The scope of the LDAP search is limited to the immediate subordinates of the base object. The LDAP search is anonymous by default. The LDAP monitor makes use of the *com.novell.ldap.LDAPConnection* class.

Monitor facts

Class Name	`org.opennms.netmgt.poller.monitors.LdapMonitor`
Remote Enabled	true

Configuration and Usage

Table 37. Monitor specific parameters for the LdapMonitor

Parameter	Description	Required	Default value
`dn`	The distinguished name to use if authenticated search is needed.	optional	`-`
`password`	The password to use if authenticated search is needed.	optional	`-`
`port`	The destination port where connection shall be attempted.	optional	`389`
`retry`	Number of attempts to get a search result.	optional	`1`
`searchbase`	The base distinguished name to search from.	optional	`base`
`searchfilter`	The LDAP search's filter.	optional	`(objectclass=*)`
`timeout`	Time in milliseconds to wait for a result from the search.	optional	`3000`
`version`	The version of the LDAP protocol to use, specified as an integer. Note: Only LDAPv3 is supported at the moment.	optional	`3`

Examples

```
<--! OpenNMS.org -->
<service name="LDAP" interval="300000" user-defined="false" status="on">
  <parameter key="port" value="389"/>
  <parameter key="version" value="3"/>
  <parameter key="searchbase" value="dc=opennms,dc=org"/>
  <parameter key="searchfilter" value="uid=ulf"/>
  <parameter key="retry" value="2"/>
  <parameter key="timeout" value="3000"/>
  <parameter key="rrd-repository" value="/var/lib/opennms/rrd/response"/>
  <parameter key="rrd-base-name" value="ldap"/>
  <parameter key="ds-name" value="ldap"/>
</service>
<monitor service="LDAP" class-name="org.opennms.netmgt.poller.monitors.LdapMonitor"/>
```

3.6.24. LdapsMonitor

The LDAPS monitor tests the response of an SSL-enabled LDAP server. The LDAPS monitor is an SSL-enabled extension of the LDAP monitor with a default TCP port value of 636. All LdapMonitor parameters apply, so please refer to LdapMonitor's documentation for more information.

Monitor facts

Class Name	`org.opennms.netmgt.poller.monitors.LdapsMonitor`
Remote Enabled	true

Configuration and Usage

Table 38. Monitor specific parameters for the LdapsMonitor

Parameter	Description	Required	Default value
`port`	The destination port where connections shall be attempted.	optional	`636`

Examples

```
<!-- LDAPS service at OpenNMS.org is on port 6636 -->
<service name="LDAPS" interval="300000" user-defined="false" status="on">
  <parameter key="port" value="6636"/>
  <parameter key="version" value="3"/>
  <parameter key="searchbase" value="dc=opennms,dc=org"/>
  <parameter key="searchfilter" value="uid=ulf"/>
  <parameter key="retry" value="2"/>
  <parameter key="timeout" value="3000"/>
  <parameter key="rrd-repository" value="/var/lib/opennms/rrd/response"/>
  <parameter key="rrd-base-name" value="ldap"/>
  <parameter key="ds-name" value="ldap"/>
</service>

<monitor service="LDAPS" class-name="org.opennms.netmgt.poller.monitors.LdapsMonitor"
/>
```

3.6.25. MemcachedMonitor

This monitor allows to monitor Memcached, a distributed memory object caching system. To monitor the service availability the monitor tests if the *Memcached* statistics can be requested. The statistics are processed and stored in RRD files. The following metrics are collected:

Table 39. Collected metrics using the MemcachedMonitor

Metric	Description
uptime	Seconds the *Memcached* server has been running since last restart.
rusageuser	User time seconds for the server process.
rusagesystem	System time seconds for the server process.
curritems	Number of items in this servers cache.
totalitems	Number of items stored on this server.

Metric	Description
bytes	Number of bytes currently used for caching items.
limitmaxbytes	Maximum configured cache size.
currconnections	Number of open connections to this *Memcached*.
totalconnections	Number of successful connect attempts to this server since start.
connectionstructure	Number of internal connection handles currently held by the server.
cmdget	Number of *GET* commands received since server startup.
cmdset	Number of *SET* commands received since server startup.
gethits	Number of successful *GET* commands (cache hits) since startup.
getmisses	Number of failed *GET* requests, because nothing was cached.
evictions	Number of objects removed from the cache to free up memory.
bytesread	Number of bytes received from the network.
byteswritten	Number of bytes send to the network.
threads	Number of threads used by this server.

Monitor facts

Class Name	`org.opennms.netmgt.poller.monitors.MemcachedMonitor`
Remote Enabled	true

Configuration and Usage

Table 40. Monitor specific parameters for the MemcachedMonitor

Parameter	Description	Required	Default value
`timeout`	Timeout in milliseconds for Memcached connection establishment.	optional	`3000`
`retry`	Number of attempts to establish the Memcached connnection.	optional	`0`
`port`	TCP port connecting to Memcached.	optional	`11211`

Examples

The following example shows a configuration in the `poller-configuration.xml`.

```
<service name="Memcached" interval="300000" user-defined="false" status="on">
  <parameter key="port" value="11211" />
  <parameter key="retry" value="2" />
  <parameter key="timeout" value="3000" />
  <parameter key="rrd-repository" value="/opt/opennms/share/rrd/response" />
  <parameter key="ds-name" value="memcached" />
  <parameter key="rrd-base-name" value="memcached" />
</service>

<monitor service="Memcached" class-name=
"org.opennms.netmgt.poller.monitors.MemcachedMonitor" />
```

3.6.26. NetScalerGroupHealthMonitor

This monitor is designed for *Citrix® NetScaler®* loadbalancing checks. It checks if more than x percent of the servers assigned to a specific group on a loadbalanced service are active. The required data is gathered via SNMP from the *NetScaler®*. The status of the servers is determined by the *NetScaler®*. The provided service it self is not part of the check. The basis of this monitor is the *SnmpMonitorStrategy*. A valid SNMP configuration in OpenNMS Horizon for the *NetScaler®* is required.

A *NetScaler®* can manage several groups of servers per application. This monitor just covers one group at a time. If there are multiple groups to check, define one monitor per group.

This monitor is not checking the loadbalanced service it self.

Monitor facts

Class Name	`org.opennms.netmgt.poller.monitors.NetScalerGroupHealthMonitor`
Remote Enabled	false

Configuration and Usage

Table 41. Monitor specific parameters for the NetScalerGroupHealthMonitor

Parameter	Description	Required	Default value
`group-name`	The name of the server group to check	required	-
`group-health`	The percentage of active servers vs total server of the group as an integer	optional	`60`

Examples

The following example checks a server group called *central_webfront_http*. If at least 70% of the

servers are active, the service is up. If less then 70% of the servers are active the service is down. A configuration like the following can be used for the example in the `poller-configuration.xml`.

```
<service name="NetScaler_Health" interval="300000" user-defined="false" status="on">
   <parameter key="group-name" value="central_webfront_http" />
   <parameter key="group-health" value="70" />
</service>

<monitor service="NetScaler_Health" class-name=
"org.opennms.netmgt.poller.monitors.NetScalerGroupHealthMonitor" />
```

Details about the used SNMP checks

The monitor checks the status of the server group based on the *NS-ROOT-MIB* using the *svcGrpMemberState*. *svcGrpMemberState* is part of the *serviceGroupMemberTable*. The *serviceGroupMemberTable* is indexed by *svcGrpMemberGroupName* and *svcGrpMemberName*. A initial lookup for the `group-name` is performed. Based on the lookup the *serviceGroupMemberTable* is walked with the numeric representation of the server group. The monitor interprets just the server status code *7-up* as active server. Other status codes like *2-unknown* or *3-busy* are counted for total amount of servers.

3.6.27. NrpeMonitor

This monitor allows to test plugins and checks running on the Nagios Remote Plugin Executor (*NRPE*) framework. The monitor allows to test the status output of any available check command executed by *NRPE*. Between OpenNMS Horizon and *Nagios* are some conceptional differences. In OpenNMS Horizon a service can only be available or not available and the response time for the service is measured. *Nagios* on the other hand combines service availability, performance data collection and thresholding in one check command. For this reason a *Nagios* check command can have more states then *OK* and *CRITICAL*. Using the *NrpeMonitor* marks all check command results other than *OK* as *down*. The full output of the check command output message is passed into the service down event in OpenNMS Horizon.

NRPE configuration on the server is required and the check command has to be configured, e.g. `command[check_apt]=/usr/lib/nagios/plugins/check_apt`

OpenNMS Horizon executes every *NRPE* check in a Java thread without *fork()* a process and it is more resource friendly. Nevertheless it is possible to run *NRPE* plugins which combine a lot of external programs like `sed`, `awk` or `cut`. Be aware, each command end up in forking additional processes.

Monitor facts

Class Name	`org.opennms.netmgt.poller.monitors.NrpeMonitor`
Remote Enabled	false

Configuration and Usage

Table 42. Monitor specific parameters for the NrpeMonitor

Parameter	Description	Required	Default value
`retry`	Number of retries before the service is marked as *down*.	optional	`0`
`timeout`	Time in milliseconds to wait for the *NRPE* executing a check command.	optional	`3000`
`command`	The {check_name} of the command configured as `command[{check_name}]="/path/to/plugin/check-script"	required	empty
`port`	Port to access *NRPE* on the remote server.	optional	`5666`
`padding`	Padding for sending the command to the *NRPE* agent.	optional	`2`
`usessl`	Enable encryption of network communication. *NRPE* uses SSL with anonymous DH and the following cipher suite *TLS_DH_anon_WITH_AES_128_CBC_SHA*	optional	`true`

Example: Using *check_apt* with *NRPE*

This examples shows how to configure the *NrpeMonitor* running the *check_apt* command on a configured *NRPE*.

Configuration of the NRPE check command on the agent in 'nrpe.cfg'

```
command[check_apt]=/usr/lib/nagios/plugins/check_apt
```

Configuration to test the NRPE plugin with the NrpeMonitor

```
<service name="NRPE-Check-APT" interval="300000" user-defined="false" status="on">
  <parameter key="retry" value="3" />
  <parameter key="timeout" value="3000" />
  <parameter key="port" value="5666" />
  <parameter key="command" value="check_apt" />
  <parameter key="padding" value="2" />
</service>

<monitor service="NRPE-Check-APT" class-name=
"org.opennms.netmgt.poller.monitors.NrpeMonitor" />
```

3.6.28. NtpMonitor

The NTP monitor tests for NTP service availability. During the poll an NTP request query packet is generated. If a response is received, it is parsed and validated. If the response is a valid NTP response, the service is considered available.

Monitor facts

Class Name	`org.opennms.netmgt.poller.monitors.NtpMonitor`
Remote Enabled	true

Configuration and Usage

Table 43. Monitor specific parameters for the NtpMonitor

Parameter	Description	Required	Default value
`port`	The destination port where the NTP request shall be sent.	optional	`123`
`retry`	Number of attempts to get a response.	optional	`0`
`timeout`	Time in milliseconds to wait for a response.	optional	`5000`

Examples

```
<--! Fast NTP server -->
<service name="NTP" interval="300000" user-defined="false" status="on">
  <parameter key="retry" value="2"/>
  <parameter key="timeout" value="1000"/>
  <parameter key="rrd-repository" value="/var/lib/opennms/rrd/response"/>
  <parameter key="rrd-base-name" value="ntp"/>
  <parameter key="ds-name" value="ntp"/>
</service>
<monitor service="NTP" class-name="org.opennms.netmgt.poller.monitors.NtpMonitor"/>
```

3.6.29. OmsaStorageMonitor

With OmsaStorageMonitor you are able to monitor your Dell OpenManaged servers RAID array status. The following OIDS from the STORAGEMANAGEMENT-MIB are supported by this monitor:

```
virtualDiskRollUpStatus                     .1.3.6.1.4.1.674.10893.1.20.140.1.1.19
arrayDiskLogicalConnectionVirtualDiskNumber .1.3.6.1.4.1.674.10893.1.20.140.3.1.5
arrayDiskNexusID                            .1.3.6.1.4.1.674.10893.1.20.130.4.1.26
arrayDiskLogicalConnectionArrayDiskNumber   .1.3.6.1.4.1.674.10893.1.20.140.3.1.3
arrayDiskState                              .1.3.6.1.4.1.674.10893.1.20.130.4.1.4
```

To test the status of the disk array the `virtualDiskRollUpStatus` is used. If the result of the `virtualDiskRollUpStatus` is not `3` the monitors is marked as *down*.

Table 44. Possible result of virtual disk rollup status

Result	State description	Monitor state in OpenNMS Horizon
1	*other*	DOWN
2	*unknown*	DOWN
3	*ok*	UP
4	*non-critical*	DOWN
5	*critical*	DOWN
6	*non-recoverable*	DOWN

You'll need to know the maximum number of possible logical disks you have in your environment. For example: If you have 3 RAID arrays, you need for each logical disk array a service poller.

To give more detailed information in case of an disk array error, the monitor tries to identify the problem using the other OIDs. This values are used to enrich the error reason in the service down event. The disk array state is resolved to a human readable value by the following status table.

Table 45. Possible array disk state errors

Value	Status
1	*Ready*
2	*Failed*
3	*Online*
4	*Offline*
6	*Degraded*
7	*Recovering*
11	*Removed*
15	*Resynching*
24	*Rebuilding*
25	*noMedia*
26	*Formating*
28	*Running Diagnostics*
35	*Initializing*

Monitor facts

Class Name	`org.opennms.netmgt.poller.monitors.OmsaStorageMonitor`

Remote Enabled	false

Configuration and Usage

Monitor specific parameters for the OmsaStorageMonitor

Parameter	Description	Required	Default value
virtualDiskNumber	The disk index of your RAID array	optional	1
retry	Amount of attempts opening a connection and try to get the greeting banner before the service goes down.	optional	from snmp-config.xml
timeout	Time in milliseconds to wait before receiving the SNMP response.	optional	from snmp-config.xml
port	The TCP port OpenManage is listening	optional	from snmp-config.xml

Examples

Some example configuration how to configure the monitor in the `poller-configuration.xml`.

The RAID array monitor for your first array is configured with `virtualDiskNumber = 1` and can look like this:

```
<service name="OMSA-Disk-Array-1" interval="300000" user-defined="false" status="on">
 <parameter key="retry" value="3"/>
 <parameter key="timeout" value="6000"/>
 <parameter key="virtualDiskNumber" value="1"/>
</service>

<monitor service="OMSA-Disk-Array-1" class-name=
"org.opennms.netmgt.poller.monitors.OmsaStorageMonitor"/>
```

If there is more than one RAID array to monitor you need an additional configuration. In this case `virtualDiskNumber = 2`. And so on...

```
<service name="OMSA-Disk-Array-2" interval="300000" user-defined="false" status="on">
 <parameter key="retry" value="3"/>
 <parameter key="timeout" value="6000"/>
 <parameter key="virtualDiskNumber" value="2"/>
</service>

<monitor service="OMSA-Disk-Array-2" class-name=
"org.opennms.netmgt.poller.monitors.OmsaStorageMonitor"/>
```

3.6.30. OpenManageChassisMonitor

The OpenManageChassis monitor tests the status of a Dell chassis by querying its SNMP agent. The monitor polls the value of the node's SNMP OID .1.3.6.1.4.1.674.10892.1.300.10.1.4.1 (MIB-Dell-10892::chassisStatus). If the value is *OK* (3), the service is considered available.

As this monitor uses SNMP, the queried nodes must have proper SNMP configuration in *snmp-config.xml*.

Monitor facts

Class Name	`org.opennms.netmgt.poller.monitors.OpenManageChassisMonitor`
Remote Enabled	false

Configuration and Usage

Table 46. Monitor specific parameters for the OpenManageChassisMonitor

Parameter	Description	Required	Default value
`port`	The port to which connection shall be tried.	optional	from `snmp-config.xml`
`retry`	Number of polls to attempt.	optional	from `snmp-config.xml`
`timeout`	Time (in milliseconds) to wait before receiving the SNMP response.	optional	from `snmp-config.xml`

Examples

```
<!-- Overriding default SNMP config -->
<service name="OMA-Chassis" interval="300000" user-defined="false" status="on">
  <parameter key="retry" value="3"/>
  <parameter key="timeout" value="5000"/>
</service>

<monitor service="OMA-Chassis" class-name=
"org.opennms.netmgt.poller.monitors.OpenManageChassisMonitor" />
```

Dell MIBs

Dell MIBs can be found here. Download the DCMIB<version>.zip or DCMIB<version>.exe file corresponding to the version of your OpenManage agents. The latest one should be good enough for all previous version though.

3.6.31. PercMonitor

This monitor tests the status of a PERC RAID array.

The monitor first polls the RAID-Adapter-MIB::logicaldriveTable (1.3.6.1.4.1.3582.1.1.2) to retrieve the status of the RAID array you want to monitor. If the value of the status object of the corresponding logicaldriveEntry is not 2, the array is degraded and the monitor further polls the RAID-Adapter-MIB::physicaldriveTable (1.3.6.1.4.1.3582.1.1.3) to detect the failed drive(s).

This monitor requires the outdated persnmpd software to be installed on the polled nodes. Please prefer using OmsaStorageMonitor monitor where possible.

Monitor facts

Class Name	`org.opennms.netmgt.poller.monitors.PercMonitor`
Remote Enabled	false (relies on SNMP configuration)

Configuration and Usage

Table 47. Monitor specific parameters for the PercMonitor

Parameter	Description	Required	Default value
`array`	The RAID array you want to monitor.	optional	`0.0`
`port`	The UDP port to connect to	optional	`from snmp-config.xml`
`retry`	The number of attempts the monitor shall try getting a response.	optional	`from snmp-config.xml`
`timeout`	The amount of thime in milliseconds the monitor shall wait for a reponse during each polling attempt.	optional	`from snmp-config.xml`

Examples

```
<!-- Monitor 1st RAID arrays using configuration from snmp-config.xml -->
<service name="PERC" interval="300000" user-defined="false" status="on" />

<monitor service="PERC" class-name="org.opennms.netmgt.poller.monitors.PercMonitor" />
```

3.6.32. Pop3Monitor

The POP3 monitor tests for POP3 service availability on a node. The monitor first tries to establish a TCP connection on the specified port. If a connection is established, a service banner should have been received. The monitor makes sure the service banner is a valid POP3 banner (ie: starts with "+OK"). If the banner is valid, the monitor sends a *QUIT* POP3 command and makes sure the service answers with a valid response (ie: a response that starts with "+OK"). The service is considered available if the service's answer to the *QUIT* command is valid.

The behaviour can be simulated with `telnet`:

```
$ telnet mail.opennms.org 110
Trying 192.168.0.100
Connected to mail.opennms.org.
Escape character is '^]'.
+OK <21860.1076718099@mail.opennms.org>
quit
+OK
Connection closed by foreign host.
```

Monitor facts

Class Name	`org.opennms.netmgt.poller.monitors.Pop3Monitor`
Remote Enabled	true

Configuration and Usage

Table 48. Monitor specific parameters for the Pop3Monitor

Parameter	Description	Required	Default value
`port`	TCP port to connect to.	optional	`110`
`retry`	Number of attempts to find the service available.	optional	`0`
`strict-timeout`	Boolean If set to `true`, makes sure that at least `timeout` milliseconds are elapsed between attempts.	optional	`false`
`timeout`	Timeout in milliseconds for the underlying socket's *connect* and *read* operations.	optional	`3000`

Examples

```
<service name="POP3" interval="300000" user-defined="false" status="on">
  <parameter key="retry" value="2"/>
  <parameter key="timeout" value="3000"/>
  <parameter key="rrd-repository" value="/var/lib/opennms/rrd/response"/>
  <parameter key="rrd-base-name" value="pop3"/>
  <parameter key="ds-name" value="pop3"/>
</service>
<monitor service="POP3" class-name="org.opennms.netmgt.poller.monitors.Pop3Monitor"/>
```

3.6.33. PrTableMonitor

The PrTableMonitor monitor tests the prTable of a net-snmp SNMP agent.

> A table containing information on running programs/daemons configured for monitoring in the snmpd.conf file of the agent. Processes violating the number of running processes required by the agent's configuration file are flagged with numerical and textual errors.
>
> — UCD-SNMP-MIB

The monitor looks up the *prErrorFlag* entries of this table. If the value of a *prErrorFlag* entry in this table is set to "1" the service is considered unavailable.

> A Error flag to indicate trouble with a process. It goes to 1 if there is an error, 0 if no error.
>
> — UCD-SNMP-MIB

Monitor facts

Class Name	`org.opennms.netmgt.poller.monitors.PrTableMonitor`
Remote Enabled	false

Configuration and Usage

Table 49. Monitor specific parameters for the PrTableMonitor

Parameter	Description	Required	Default value
`port`	The port to which connection shall be tried.	optional	from `snmp-config.xml`
`retry`	Number of polls to attempt.	optional	from `snmp-config.xml`
`retries`	**Deprecated**. Same as `retry`. Parameter `retry` takes precedence if both are set.	optional	from `snmp-config.xml`
`timeout`	Time in milliseconds to wait before receiving the SNMP response.	optional	from `snmp-config.xml`

Examples

```
<!-- Overriding default SNMP config -->
<service name="Process-Table" interval="300000" user-defined="false" status="on">
  <parameter key="retry" value="3"/>
  <parameter key="timeout" value="5000"/>
</service>

<monitor service="Process-Table" class-name=
"org.opennms.netmgt.poller.monitors.PrTableMonitor" />
```

UCD-SNMP-MIB

The UCD-SNMP-MIB may be found here.

3.6.34. RadiusAuthMonitor

This monitor allows to test the functionality of the RADIUS authentication system. The availability is tested by sending an *AUTH* packet to the *RADIUS server*. If a valid *ACCEPT* response is received, the *RADIUS* service is *up* and considered as available.

To use this monitor it is required to install the *RADIUS* protocol for OpenNMS Horizon.

```
{apt-get,yum} install {opennms-package-base-name}-plugin-protocol-radius
```

The test is similar to test the behavior of a *RADIUS* server by evaluating the result with the command line tool `radtest`.

```
root@vagrant:~# radtest "John Doe" hello 127.0.0.1 1812 radiuspassword
Sending Access-Request of id 49 to 127.0.0.1 port 1812
    User-Name = "John Doe"
    User-Password = "hello"
    NAS-IP-Address = 127.0.0.1
    NAS-Port = 1812
    Message-Authenticator = 0x00000000000000000000000000000000
rad_recv: Access-Accept packet from host 127.0.0.1 port 1812, id=49, length=37 ①
    Reply-Message = "Hello, John Doe"
```

① The `Access-Accept` message which is evaluated by the monitor.

Monitor facts

Class Name	`org.opennms.protocols.radius.monitor.RadiusAuthMonitor`
Remote Enabled	false

Configuration and Usage

Table 50. Monitor specific parameters for the RadiusAuthMonitor

Parameter	Description	Required	Default value
`timeout`	Time in milliseconds to wait for the *RADIUS* service.	optional	`5000`
`retry`	This is a placeholder for the second optional monitor parameter description.	optional	`0`
`authport`	*RADIUS* authentication port.	optional	`1812`
`acctport`	*RADIUS* accounting port.	optional	`1813`
`user`	Username to test the authentication	optional	`OpenNMS`
`password`	Password to test the authentication	optional	`OpenNMS`
`secret`	The *RADIUS* shared secret used for communication between the *client/NAS* and the *RADIUS* server.	optional	`secret`
`authtype`	*RADIUS* authentication type. The following authentication types are supported: `chap`, `pap`, `mschapv1`, `mschapv2`, `eapmd5`, `eapmschapv2`	optional	`pap`
`nasid`	The Network Access Server identifier originating the *Access-Request*.	optional	`opennms`

Examples

Example configuration how to configure the monitor in the `poller-configuration.xml`.

```
<service name="Radius-Authentication" interval="300000" user-defined="false" status=
"on">
  <parameter key="retry" value="3" />
  <parameter key="timeout" value="3000" />
  <parameter key="user" value="John Doe" />
  <parameter key="password" value="hello" />
  <parameter key="secret" value="radiuspassword" />
  <parameter key="rrd-repository" value="/var/lib/opennms/rrd/response" />
  <parameter key="ds-name" value="radiusauth" />
</service>

<monitor service="Radius-Authentication" class-name=
"org.opennms.protocols.radius.monitor.RadiusAuthMonitor" />
```

3.6.35. SmbMonitor

This monitor is used to test the *NetBIOS over TCP/IP* name resolution in Microsoft Windows environments. The monitor tries to retrieve a *NetBIOS name* for the IP address of the interface. Name services for *NetBIOS* in Microsoft Windows are provided on port 137/UDP or 137/TCP.

The service uses the IP address of the interface, where the monitor is assigned to. The service is *up* if for the given IP address a *NetBIOS name* is registered and can be resolved.

For troubleshooting see the usage of the Microsoft Windows command line tool `nbtstat` or on Linux `nmblookup`.

Microsoft deprecated the usage of *NetBIOS*. Since Windows Server 2000 *DNS* is used as the default name resolution.

Monitor facts

Class Name	`org.opennms.netmgt.poller.monitors.SmbMonitor`
Remote Enabled	false

Configuration and Usage

Table 51. Monitor specific parameters for the SmbMonitor

Parameter	Description	Required	Default value
`retry`	Number of attempts to get a valid response	required	`-`
`timeout`	Timeout in milliseconds for TCP connection establishment	required	`-`
`do-node-status`	Try to get the NetBIOS node status type for the given address	optional	`true`

Examples

Some example configuration how to configure the monitor in the `poller-configuration.xml`.

```
<service name="SMB" interval="300000" user-defined="false" status="on">
  <parameter key="retry" value="1"/>
  <parameter key="timeout" value="3000"/>
</service>

<monitor service="SMB" class-name="org.opennms.netmgt.poller.monitors.SmbMonitor"/>
```

3.6.36. SnmpMonitor

The SNMP monitor gives a generic possibility to monitor states and results from SNMP agents. This monitor has two basic operation modes:

- Test the response value of one specific *OID* (scalar object identifier);
- Test multiple values in a whole *table*.

To decide which mode should be used, the `walk` and `match-all` parameters are used.

See the `Operating mode selection'' and` Monitor specific parameters for the SnmpMonitor" tables below for more information about these operation modes.

Table 52. Operating mode selection

walk	match-all	Operating mode
`true`	`true`	tabular, all values must match
	`false`	tabular, any value must match
	`count`	specifies that the value of at least minimum and at most maximum objects encountered in
`false`	`true`	scalar
	`false`	scalar
	`count`	tabular, between `minimum` and `maximum` values must match

This monitor can't be used on the OpenNMS Horizon Remote Poller. It is currently not possible for the Remote Poller to have access to the SNMP configuration of a central OpenNMS Horizon.

Monitor facts

Class Name	`org.opennms.netmgt.poller.monitors.SnmpMonitor`
Remote Enabled	false

Configuration and Usage

Table 53. Monitor specific parameters for the SnmpMonitor

Parameter	Description	Required	Default value
`hex`	Specifies that the value monitored should be compared against its hexadecimal representation. Useful when the monitored value is a string containing non-printable characters.	optional	`false`

Parameter	Description	Required	Default value
`match-all`	Can be set to: `count`: specifies that the value of at least minimum and at most maximum objects encountered in the walk must match the criteria specified by `operand` and `operator`. `true` and `walk` is set to `true`: specifies that the value of **every** object encountered in the walk must match the criteria specified by the `operand` and `operator` parameters. `false` and `walk` is set to `true`: specifies that the value of **any** object encountered in the walk must match the criteria specified by the `operand` and `operator` parameters.	optional	`true`
`maximum`	Valid only when `match-all` is set to `count`, otherwise ignored. Should be used in conjunction with the `minimum` parameter. Specifies that the value of *at most* `maximum` objects encountered in the walk must meet the criteria specified by the `operand` and `operator` parameters.	optional	`0`
`minimum`	Valid only when `match-all` is set to `count`, otherwise ignored. Should be used in conjunction with the `maximum` parameter. Specifies that the value of *at least* `minimum` objects encountered in the walk must meet the criteria specified by the `operand` and `operator` parameters.	optional	`0`
`oid`	The object identifier of the *MIB* object to monitor. If no other parameters are present, the monitor asserts that the agent's response for this object must include a valid value (as opposed to an error, no-such-name, or end-of-view condition) that is non-null.	optional	`.1.3.6.1.2.1.1.2.0` (SNMPv2-MIB::SysObjectID)
`operand`	The value to be compared against the observed value of the monitored object. Note: Comparison will always succeed if either the `operand` or `operator` parameter isn't set and the monitored value is non-null.	optional	`-`

Parameter	Description	Required	Default value
operator	The operator to be used for comparing the monitored object against the operand parameter. Must be one of the following symbolic operators: < (<): Less than. Both operand and observed object value must be numeric. > (>): Greater than. Both operand and observed object value must be numeric. <= (⇐): Less than or equal to. Both operand and observed object value must be numeric. >= (>=): Greater than or equal to. Both operand and observed object value must be numeric. =: Equal to. Applied in numeric context if both operand and observed object value are numeric, otherwise in string context as a case-sensitive exact match. !=: Not equal to. Applied in numeric context if both operand and observed object value are numeric, otherwise in string context as a case-sensitive exact match. ~: Regular expression match. Always applied in string context. Note: Comparison will always succeed if either the operand or operator parameter isn't set and the monitored value is non-null. Keep in mind that you need to escape all < and > characters as XML entities (< and > respectively)	optional	-
port	Destination port where the SNMP requests shall be sent.	optional	from snmp-config.xml
reason-template	A user-provided template used for the monitor's reason code if the service is unavailable. Defaults to a reasonable value if unset. See below for an explanation of the possible template parameters.	optional	depends on operation mode
retry	Number of polls to attempt.	optional	from snmp-config.xml
retries	**Deprecated** Same as retry. Parameter retry takes precedence if both are set.	optional	from snmp-config.xml
timeout	Timeout in milliseconds for retrieving the object's value.	optional	from snmp-config.xml

Parameter	Description	Required	Default value
walk	false: Sets the monitor to poll for a scalar object unless if the match-all parameter is set to count, in which case the match-all parameter takes precedence. true: Sets the monitor to poll for a tabular object where the match-all parameter defines how the tabular object's values must match the criteria defined by the operator and operand parameters. See also the match-all, minimum, and maximum parameters.	optional	false

Table 54. Variables which can be used in the reason-template parameter

Variable	Description
${hex}	Value of the hex parameter.
${ipaddr}	IP address polled.
${matchAll}	Value of the match-all parameter.
${matchCount}	When match-all is set to count, contains the number of matching instances encountered.
${maximum}	Value of the maximum parameter.
${minimum}	Value of the minimum paramater.
${observedValue}	Polled value that made the monitor succeed or fail.
${oid}	Value of the oid parameter.
${operand}	Value of the operand parameter.
${operator}	Value of the operator parameter.
${port}	Value of the port parameter.
${retry}	Value of the retry parameter.
${timeout}	Value of the timeout parameter.
${walk}	Value of the walk parameter.

Example for monitoring scalar object

As a working example we want to monitor the thermal system fan status which is provided as a scalar object ID.

```
cpqHeThermalSystemFanStatus .1.3.6.1.4.1.232.6.2.6.4.0
```

The manufacturer *MIB* gives the following information:

Description of the cpqHeThermalSystemFanStatus from CPQHLTH-MIB

```
SYNTAX  INTEGER  {
    other    (1),
    ok       (2),
    degraded (3),
    failed   (4)
}
ACCESS  read-only
DESCRIPTION
"The status of the fan(s) in the system.

This value will be one of the following:
other(1)
Fan status detection is not supported by this system or driver.

ok(2)
All fans are operating properly.

degraded(3)
A non-required fan is not operating properly.

failed(4)
A required fan is not operating properly.

If the cpqHeThermalDegradedAction is set to shutdown(3) the
system will be shutdown if the failed(4) condition occurs."
```

The SnmpMonitor is configured to test if the fan status returns *ok(2)*. If so, the service is marked as *up*. Any other value indicates a problem with the thermal fan status and marks the service *down*.

Example SnmpMonitor as HP InsightManager fan monitor in poller-configuration.xml

```
<service name="HP-Insight-Fan-System" interval="300000" user-defined="false" status=
"on">
    <parameter key="oid" value=".1.3.6.1.4.1.232.6.2.6.4.0"/> ①
    <parameter key="operator" value="="/> ②
    <parameter key="operand" value="2"/> ③
    <parameter key="reason-template" value="System fan status is not ok. The state
should be ok(${operand}) the observed value is ${observedValue}. Please check your HP
Insight Manager. Syntax: other(1), ok(2), degraded(3), failed(4)"/> ④
</service>

<monitor service="HP-Insight-Fan-System" class-name=
"org.opennms.netmgt.poller.monitors.SnmpMonitor" />
```

① Scalar object ID to test

② Operator for testing the response value

③ Integer 2 as operand for the test

④ Encode *MIB* status in the reason code to give more detailed information if the service goes down

Example test SNMP table with all matching values

The second mode shows how to monitor values of a whole SNMP table. As a practical use case the status of a set of physical drives is monitored. This example configuration shows the status monitoring from the CPQIDA-MIB.

We use as a scalar object id the physical drive status given by the following tabular OID:

```
cpqDaPhyDrvStatus .1.3.6.1.4.1.232.3.2.5.1.1.6
```

Description of the cpqDaPhyDrvStatus object id from CPQIDA-MIB

```
SYNTAX  INTEGER  {
    other             (1),
    ok                (2),
    failed            (3),
    predictiveFailure (4)
}
ACCESS  read-only
DESCRIPTION
Physical Drive Status.
This shows the status of the physical drive.
The following values are valid for the physical drive status:

other (1)
 Indicates that the instrument agent does not recognize
 the drive.  You may need to upgrade your instrument agent
 and/or driver software.

ok (2)
 Indicates the drive is functioning properly.

failed (3)
 Indicates that the drive is no longer operating and
 should be replaced.

predictiveFailure(4)
 Indicates that the drive has a predictive failure error and
 should be replaced.
```

The configuration in our monitor will test all physical drives for status *ok(2)*.

Example SnmpMonitor as HP Insight physical drive monitor in poller-configuration.xml

```
<service name="HP-Insight-Drive-Physical" interval="300000" user-defined="false"
status="on">
    <parameter key="oid" value=".1.3.6.1.4.1.232.3.2.5.1.1.6"/> ①
    <parameter key="walk" value="true"/> ②
    <parameter key="operator" value="="/> ③
    <parameter key="operand" value="2"/> ④
    <parameter key="match-all" value="true"/> ⑤
    <parameter key="reason-template" value="One or more physical drives are not ok.
The state should be ok(${operand}) the observed value is ${observedValue}. Please
check your HP Insight Manager. Syntax: other(1), ok(2), failed(3),
predictiveFailure(4), erasing(5), eraseDone(6), eraseQueued(7)"/> ⑥
</service>

<monitor service="HP-Insight-Drive-Physical" class-name=
"org.opennms.netmgt.poller.monitors.SnmpMonitor" />
```

① OID for SNMP table with all physical drive states

② Enable *walk mode* to test every entry in the table against the test criteria

③ Test operator for integer

④ Integer 2 as operand for the test

⑤ Test in *walk mode* has to be passed for every entry in the table

⑥ Encode *MIB* status in the reason code to give more detailed information if the service goes down

Example test SNMP table with all matching values

This example shows how to use the SnmpMonitor to test if the number of static routes are within a given boundary. The service is marked as *up* if at least 3 and at maxium 10 static routes are set on a network device. This status can be monitored by polling the table *ipRouteProto* from the RFC1213-MIB2.

```
ipRouteProto 1.3.6.1.2.1.4.21.1.9
```

The *MIB* description gives us the following information:

```
SYNTAX  INTEGER  {
    other(1),
    local(2),
    netmgmt(3),
    icmp(4),
    egp(5),
    ggp(6),
    hello(7),
    rip(8),
    is-is(9),
    es-is(10),
    ciscoIgrp(11),
    bbnSpfIgp(12),
    ospf(13),
    bgp(14)}
}
ACCESS  read-only
DESCRIPTION
"The routing mechanism via which this route was learned.
Inclusion of values for gateway routing protocols is not
intended to imply that hosts should support those protocols."
```

To monitor only local routes, the test should be applied only on entries in the *ipRouteProto* table with value 2. The number of entries in the whole *ipRouteProto* table has to be counted and the boundaries on the number has to be applied.

Example SnmpMonitor used to test if the number of local static route entries are between 3 or 10.

```
<service name="All-Static-Routes" interval="300000" user-defined="false" status="on">
 <parameter key="oid" value=".1.3.6.1.2.1.4.21.1.9" /> ①
 <parameter key="walk" value="true" /> ②
 <parameter key="operator" value="=" /> ③
 <parameter key="operand" value="2" /> ④
 <parameter key="match-all" value="count" /> ⑤
 <parameter key="minimum" value="3" /> ⑥
 <parameter key="maximum" value="10" /> ⑦
</service>

<monitor service="All-Static-Routes" class-name=
"org.opennms.netmgt.poller.monitors.SnmpMonitor" />
```

① OID for SNMP table *ipRouteProto*

② Enable *walk mode* to test every entry in the table against the test criteria

③ Test operator for integer

④ Integer 2 as operand for testing local route entries

⑤ Test in *walk mode* has is set to `count` to get the number of entries in the table regarding `operator` and `operand`

⑥ Lower count boundary set to `3`

⑦ High count boundary is set to `10`

3.6.37. SshMonitor

The SSH monitor tests the availability of a SSH service. During the poll an attempt is made to connect on the specified port. If the connection request is successful, then the service is considered up. Optionaly, the banner line generated by the service may be parsed and compared against a pattern before the service is considered up.

Monitor facts

Class Name	`org.opennms.netmgt.poller.monitors.SshMonitor`
Remote Enabled	true

Configuration and Usage

Table 55. Monitor specific parameters for the SshMonitor

Parameter	Description	Required	Default value
`banner`	Regular expression to be matched against the service's banner.	optional	`-`
`client-banner`	The client banner that OpenNMS Horizon will use to identify itself on the service.	optional	`SSH-1.99-OpenNMS_1.5`
`match`	Regular expression to be matched against the service's banner. Deprecated, please use the `banner` parameter instead. Note that this parameter takes precedence over the `banner` parameter, though.	optional	`-`
`port`	TCP port to which SSH connection shall be tried.	optional	`22`
`retry`	Number of attempts to establish the SSH connnection.	optional	`0`
`timeout`	Timeout in milliseconds for SSH connection establishment.	optional	`3000`

Examples

```
<service name="SSH" interval="300000" user-defined="false" status="on">
  <parameter key="retry" value="1"/>
  <parameter key="banner" value="SSH"/>
  <parameter key="client-banner" value="OpenNMS poller"/>
  <parameter key="timeout" value="5000"/>
  <parameter key="rrd-repository" value="/var/lib/opennms/rrd/response"/>
  <parameter key="rrd-base-name" value="ssh"/>
  <parameter key="ds-name" value="ssh"/>
</service>
<monitor service="SSH" class-name="org.opennms.netmgt.poller.monitors.SshMonitor"/>
```

3.6.38. SSLCertMonitor

This monitor is used to test if a SSL certificate presented by a remote network server are valid. A certificate is invalid if its initial time is prior to the current time, or if the current time is prior to 7 days (configurable) before the expiration time. The monitor only supports SSL on the socket and does not support a higher level protocol above it. Additionally, it does not support Server Name Indication (SNI) and so is unable to validate different certificates if they would be presented on the same connection.

You can simulate the behavior by running a command like this:

```
echo | openssl s_client -connect <site>:<port> 2>/dev/null | openssl x509 -noout
-dates
```

The output shows you the time range a certificate is valid:

```
notBefore=Dec 24 14:11:34 2013 GMT
notAfter=Dec 25 10:37:40 2014 GMT
```

You can configure a threshold in days applied on the `notAfter` date.

Monitor facts

Class Name	`org.opennms.netmgt.poller.monitors.SSLCertMonitor`
Remote Enabled	true

Configuration and Usage

Table 56. Monitor specific parameters for the SSLCertMonitor

Parameter	Description	Required	Default value
`port`	TCP port for the service with SSL certificate.	required	`-1`

Parameter	Description	Required	Default value
`retry`	Number of attempts to get the certificate state	optional	`0`
`timeout`	Time in milliseconds to wait before next attempt.	optional	`3000`
`days`	Number of days before the certificate expires that we mark the service as failed.	optional	`7`

The monitor has no support for communicating on other protocol layers above the SSL session layer. It is not able to send a Host header for HTTPS, or issue a STARTTLS command for IMAP, POP3, SMTP, FTP, XMPP, LDAP, or NNTP.

Examples

The following example shows how to monitor SSL certificates on services like IMAPS, SMTPS and HTTPS. If the certificates expire within 30 days the service goes down and indicates this issue in the reason of the monitor. In this example the monitoring interval is reduced to test the certificate every 2 hours (7,200,000 ms). Configuration in `poller-configuration.xml` is as the following:

```
<service name="SSL-Cert-IMAPS-993" interval="7200000" user-defined="false" status="on
">
    <parameter key="retry" value="2"/>
    <parameter key="timeout" value="2000"/>
    <parameter key="port" value="993"/>
    <parameter key="days" value="30"/>
</service>
<service name="SSL-Cert-SMTPS-465" interval="7200000" user-defined="false" status="on
">
    <parameter key="retry" value="2"/>
    <parameter key="timeout" value="2000"/>
    <parameter key="port" value="465"/>
    <parameter key="days" value="30"/>
</service>
<service name="SSL-Cert-HTTPS-443" interval="7200000" user-defined="false" status="on
">
    <parameter key="retry" value="2"/>
    <parameter key="timeout" value="3000"/>
    <parameter key="port" value="443"/>
    <parameter key="days" value="30"/>
</service>

<monitor service="SSL-Cert-IMAPS-993" class-name=
"org.opennms.netmgt.poller.monitors.SSLCertMonitor" />
<monitor service="SSL-Cert-SMTPS-465" class-name=
"org.opennms.netmgt.poller.monitors.SSLCertMonitor" />
<monitor service="SSL-Cert-HTTPS-443" class-name=
"org.opennms.netmgt.poller.monitors.SSLCertMonitor" />
```

3.6.39. StrafePingMonitor

This monitor is used to monitor packet delay variation to a specific endpoint using *ICMP*. The main use case is to monitor a *WAN* end point and visualize packet loss and *ICMP* packet round trip time deviation. The *StrafePingMonitor* performs multiple *ICMP echo requests* (ping) and stores the response-time of each as well as the packet loss, in a *RRD* file. Credit is due to Tobias Oetiker, as this graphing feature is an adaptation of the SmokePing tool that he developed.

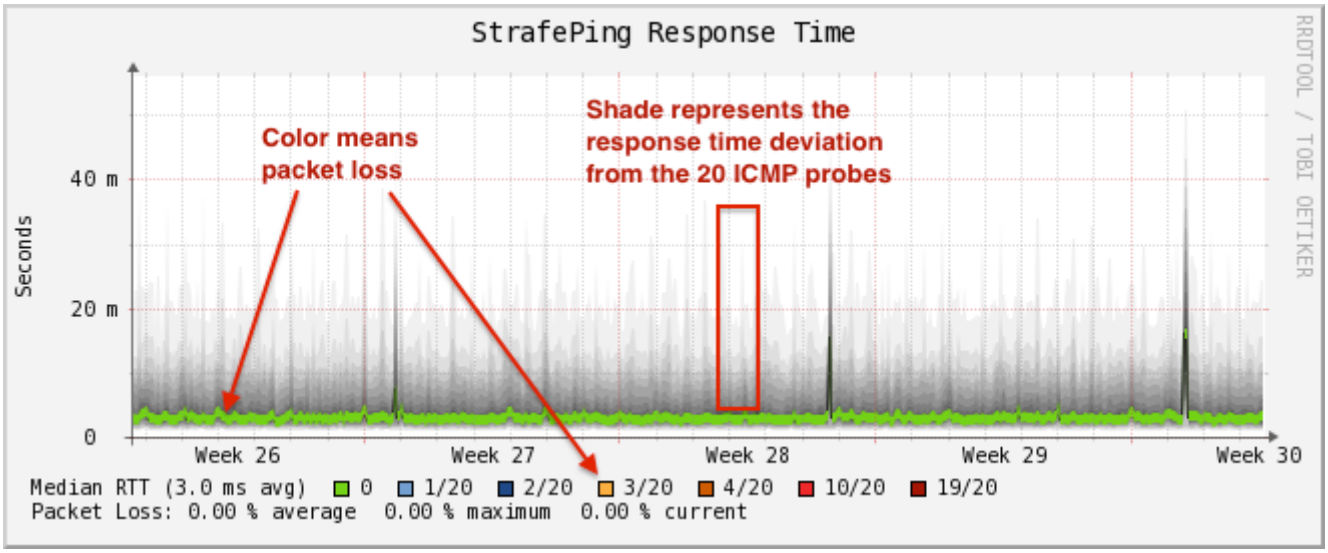


Figure 16. Visualization of a graph from the StrafePingMonitor

Monitor facts

Class Name	`org.opennms.netmgt.poller.monitors.StrafePingMonitor`
Remote Enabled	false

Configuration and Usage

Monitor specific parameters for the StrafePingMonitor

Parameter	Description	Required	Default value
`timeout`	Time in milliseconds to wait before assuming that a packet has not responded	optional	`800`
`retry`	The number of retries to attempt when a packet fails to respond in the given timeout	optional	`2`
`ping-count`	The number of pings to attempt each interval	required	`20`
`failure-ping-count`	The number of pings that need to fail for the service to be considered down	required	`20`
`wait-interval`	Time in milliseconds to wait between each *ICMP echo-request* packet	required	`50`
`rrd-repository`	The location to write *RRD data*. Generally, you will not want to change this from default	required	`$OPENNMS_HOME/share/rrd/response`
`rrd-base-name`	The name of the RRD file to write (minus the extension, `.rrd` or `.jrb`)	required	`strafeping`

Examples

The *StrafePingMonitor* is typically used on WAN connections and not activated for every ICMP enabled device in your network. Further this monitor is much I/O heavier than just a simple RRD

graph with a single ICMP response time measurement. By default you can find a separate *poller package* in the 'poller-configuration.xml' called *strafer*. Configure the `include-range` or a `filter` to enable monitoring for devices with the service *StrafePing*.

Don't forget to assign the service *StrafePing* on the IP interface to be activated.

The following example enables the monitoring for the service *StrafePing* on IP interfaces in the range 10.0.0.1 until 10.0.0.20. Additionally the Nodes have to be in a *surveillance category* named `Latency`.

```
<package name="strafer" >
   <filter>categoryName == 'Latency'</filter>
   <include-range begin="10.0.0.1" end="10.0.0.20"/>
   <rrd step="300">
     <rra>RRA:AVERAGE:0.5:1:2016</rra>
     <rra>RRA:AVERAGE:0.5:12:1488</rra>
     <rra>RRA:AVERAGE:0.5:288:366</rra>
     <rra>RRA:MAX:0.5:288:366</rra>
     <rra>RRA:MIN:0.5:288:366</rra>
   </rrd>
   <service name="StrafePing" interval="300000" user-defined="false" status="on">
     <parameter key="retry" value="0"/>
     <parameter key="timeout" value="3000"/>
     <parameter key="ping-count" value="20"/>
     <parameter key="failure-ping-count" value="20"/>
     <parameter key="wait-interval" value="50"/>
     <parameter key="rrd-repository" value="/opt/opennms/share/rrd/response"/>
     <parameter key="rrd-base-name" value="strafeping"/>
   </service>
   <downtime interval="30000" begin="0" end="300000"/>
   <downtime interval="300000" begin="300000" end="43200000"/>
   <downtime interval="600000" begin="43200000" end="432000000"/>
   <downtime begin="432000000" delete="true"/>
 </package>
 <monitor service="StrafePing" class-name=
"org.opennms.netmgt.poller.monitors.StrafePingMonitor"/>
```

3.6.40. TcpMonitor

This monitor is used to test IP Layer 4 connectivity using TCP. The monitor establishes an TCP connection to a specific port. To test the availability of the service, the greetings banner of the application is evaluated. The behavior is similar to a simple test using the `telnet` command as shown in the example.

Simulating behavior of the monitor with `telnet`

```
root@vagrant:~# telnet 127.0.0.1 22
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
SSH-2.0-OpenSSH_6.6.1p1 Ubuntu-2ubuntu2 ①
```

① Service greeting banner

Monitor facts

Class Name	`org.opennms.netmgt.poller.monitors.TcpMonitor`
Remote Enabled	true

Configuration and Usage

Table 57. Monitor specific parameters for the TcpMonitor

Parameter	Description	Required	Default value
`port`	TCP port of the application.	required	`-1`
`retry`	Number of retries before the service is marked as *down*.	optional	`0`
`timeout`	Time in milliseconds to wait for the TCP service.	optional	`3000`
`banner`	Evaluation of the service connection banner with regular expression. By default any banner result is valid.	optional	`*`

Examples

This example shows to test if the ICA service is available on TCP port 1494. The test evaluates the connection banner starting with `ICA`.

```
<service name="TCP-Citrix-ICA" interval="300000" user-defined="false" status="on">
  <parameter key="retry" value="0" />
  <parameter key="banner" value="ICA" />
  <parameter key="port" value="1494" />
  <parameter key="timeout" value="3000" />
  <parameter key="rrd-repository" value="/var/lib/opennms/rrd/response" />
  <parameter key="rrd-base-name" value="tcpCitrixIca" />
  <parameter key="ds-name" value="tcpCitrixIca" />
</service>

<monitor service="TCP-Citrix-ICA" class-name=
"org.opennms.netmgt.poller.monitors.TcpMonitor" />
```

3.6.41. SystemExecuteMonitor

If it is required to execute a system call or run a script to determine a service status, the SystemExecuteMonitor can be used. It is calling a script or system command, if required it provides additional arguments to the call. To determine the status of the service the SystemExecuteMonitor can rely on 0 or a non-0 exit code of system call. As an alternative, the output of the system call can be matched against a banner. If the banner is part of the output the status is interpreted as up. If the banner is not available in the output the status is determined as down.

Monitor facts

Class Name	`org.opennms.netmgt.poller.monitors.SystemExecuteMonitor`
Remote Enabled	true

Configuration and Usage

Table 58. Monitor specific parameters for the SystemExecuteMonitor

Parameter	Description	Required	Default value
`script`	The system-call to execute.	required	-
`args`	The arguments to hand over to the system-call. It supports variable replacement, see below.	optional	-
`banner`	A string that is match against the output of the system-call. If the output contains the banner, the service is determined as *UP*.	optional	-

The parameter `args` supports variable replacement for the following set of variables.

Table 59. Variables which can be used in the configuration

Variable	Description
`${timeout}`	Timeout in milliseconds, based on config of the service.

Variable	Description
`${timeoutsec}`	Timeout in seconds, based on config of the service.
`${retry}`	Amount of retries based on config of the service.
`${svcname}`	Service name based on the config of the service.
`${ipaddr}`	IP-address of the interface the service is bound to.
`${nodeid}`	Nodeid of the node the monitor is associated to.
`${nodelabel}`	Nodelabel of the node the monitor is associated to.

Examples

```
<parameter key="args" value="-i ${ipaddr} -t ${timeout}"/>
<parameter key="args" value="http://${nodelabel}/${svcname}/static"/>
```

SystemExecuteMonitor vs GpMonitor

The SystemExecuteMonitor is the successor of the GpMonitor. The main differences are:

- Variable replacement for the parameter args
- There are no fixed arguments handed to the system-call
- The *SystemExecuteMonitor* supports *RemotePoller* deployment

To migrate services from the *GpMonitor* to the *SystemExecuteMonitor* it is required to alter the parameter args. To match the arguments called `hoption` for the `hostAddress` and `toption` for the `timeoutInSeconds`. The args string that matches the *GpMonitor* call looks like this:

```
<parameter key="args" value="--hostname ${ipaddr} --timeout ${timeoutsec}" />
```

To migrate the GpMonitor parameters `hoption` and `toption` just replace the `--hostname` and `--timeout` directly in the `args` key.

3.6.42. VmwareCimMonitor

This monitor is part of the VMware integration provided in *Provisiond*. The monitor is specialized to test the health status provided from all *Host System* (host) sensor data.

This monitor is only executed if the host is in power state *on*.

This monitor requires to import hosts with *Provisiond* and the *VMware* import. OpenNMS Horizon requires network access to *VMware vCenter* and the hosts. To get the sensor data the credentials from *vmware-config.xml* for the responsible *vCenter* is used. The following asset fields are filled from *Provisiond* and is provided by *VMware* import feature: *VMware Management Server*, *VMware Managed Entity Type* and the *foreignId* which contains an internal *VMware vCenter Identifier*.

The global health status is evaluated by testing all available host sensors and evaluating the state of each sensor. A sensor state could be represented as the following:

- *Unknown(0)*
- *OK(5)*
- *Degraded/Warning(10)*
- *Minor failure(15)*
- *Major failure(20)*
- *Critical failure(25)*
- *Non-recoverable error(30)*

The service is *up* if **all** sensors have the status *OK(5)*. If any sensor gives another status then *OK(5)* the service is marked as *down*. The monitor error reason contains a list of all sensors which not returned status *OK(5)*.

In case of using Distributed Power Management the *standBy* state forces a service *down*. The health status is gathrered with a direct connection to the host and in stand by this connection is unavailable and the service is *down*. To deal with stand by states, the configuration *ignoreStandBy* can be used. In case of a stand by state, the service is considered as *up*.

state can be changed see the `ignoreStandBy` configuration parameter.

Monitor facts

Class Name	`org.opennms.netmgt.poller.monitors.VmwareCimMonitor`
Remote Enabled	false

Configuration and Usage

Table 60. Monitor specific parameters for the VmwareCimMonitor

Parameter	Description	Required	Default value
`retry`	Number of retries before the service is marked as down.	optional	`0`
`timeout`	Time in milliseconds to wait collecting the *CIM* sensor data.	optional	`3000`

Parameter	Description	Required	Default value
`ignoreStandBy`	Treat power state *standBy* as *up*.	optional	`false`

Examples

Some example configuration how to configure the monitor in the `poller-configuration.xml`.

```
<service name="VMwareCim-HostSystem" interval="300000" user-defined="false" status="
on">
  <parameter key="retry" value="2"/>
  <parameter key="timeout" value="3000"/>
</service>

<monitor service="VMwareCim-HostSystem" class-name=
"org.opennms.netmgt.poller.monitors.VmwareCimMonitor"/>
```

3.6.43. VmwareMonitor

This monitor is part of the VMware integration provided in *Provisiond* and test the power state of a virtual machine (VM) or a host system (host). If the power state of a VM or host is *poweredOn* the service is *up*. The state *off* the service on the VM or Host is marked as *down*. By default *standBy* is also considered as *down*. In case of using Distributed Power Management the *standBy* state can be changed see the `ignoreStandBy` configuration parameter.

The information for the status of a virtual machine is collected from the responsible *VMware vCenter* using the credentials from the *vmware-config.xml*. It is also required to get specific asset fields assigned to an imported virtual machine and host system. The following asset fields are required, which are populated by the *VMware* integration in *Provisiond*: *VMware Management Server*, *VMware Managed Entity Type* and the *foreignId* which contains an internal *VMware vCenter Identifier*.

Monitor facts

Class Name	`org.opennms.netmgt.poller.monitors.VmwareMonitor`
Remote Enabled	false

Configuration and Usage

Table 61. Monitor specific parameters for the VmwareMonitor

Parameter	Description	Required	Default value
`retry`	Number of retries before the service is marked as *down*.	optional	`0`

Parameter	Description	Required	Default value
timeout	Time in milliseconds to wait for the *vCenter* to get the power state information.	optional	3000
ignoreStandBy	Treat power state *standBy* as *up*.	optional	false

Examples

Some example configuration how to configure the monitor in the `poller-configuration.xml`.

```
<service name="VMware-ManagedEntity" interval="300000" user-defined="false" status="
on">
  <parameter key="retry" value="2"/>
  <parameter key="timeout" value="3000"/>
</service>

<monitor service="VMware-ManagedEntity" class-name=
"org.opennms.netmgt.poller.monitors.VmwareMonitor"/>
```

3.6.44. Win32ServiceMonitor

The Win32ServiceMonitor enables OpenNMS Horizon to monitor the running state of any Windows service. The service status is monitored using the Microsoft Windows® provided SNMP agent providing the LAN Manager MIB-II. For this reason it is required the SNMP agent and OpenNMS Horizon is correctly configured to allow queries against part of the *MIB* tree. The status of the service is monitored by polling the

```
svSvcOperatingState = 1.3.6.1.4.1.77.1.2.3.1.3
```

of a given service by the display name.

Monitor facts

Class Name	org.opennms.netmgt.poller.monitors.Win32ServiceMonitor
Remote Enabled	false

Configuration and Usage

Table 62. Monitor specific parameters for the Win32ServiceMonitor

Parameter	Description	Required	Default value
retry	Number of attempts to get the service state from SNMP agent	required	From snmp-config.xml

Parameter	Description	Required	Default value
`timeout`	Time in milliseconds to wait for the SNMP result before next attempt.	required	From `snmp-config.xml`
`service-name`	The name of the service, this should be the exact name of the Windows service to monitor as it appears in the Services *MSC snap-in*. Short names such as you might use with net start will **not** work here.	required	`Server`

Non-English Windows The `service-name` is sometime encoded in languages other than English. Like in French, the *Task Scheduler* service is *Planificateur de tâche*. Because of the "â" (non-English character), the OID value is encoded in hexa (0x50 6C 61 6E 69 66 69 63 61 74 65 75 72 20 64 65 20 74 C3 A2 63 68 65 73).

Troubleshooting

If you've created a *Win32ServiceMonitor* poller and are having difficulties with it not being monitored properly on your hosts, chances are there is a difference in the name of the service you've created, and the actual name in the registry.

For example, I need to monitor a process called *Example Service* on one of our production servers. I retrieve the *Display name* from looking at the service in service manager, and create an entry in the `poller-configuration.xml` files using the exact name in the *Display name* field.

However, what I don't see is the errant space at the end of the service display name that is revealed when doing the following:

```
snmpwalk -v 2c -c <communitystring> <hostname> .1.3.6.1.4.1.77.1.2.3.1.1
```

This provides the critical piece of information I am missing:

```
iso.3.6.1.4.1.77.1.2.3.1.1.31.83.116.97.102.102.119.97.114.101.32.83.84.65.70.70.86.73
.69.87.32.66.97.99.107.103.114.111.117.110.100.32 = STRING: "Example Service "
```

Note the extra space before the close quote.

The extra space at the end of the name was difficult to notice in the service manager GUI, but is easily visible in the `snmpwalk` output. The right way to fix this would be to correct the service *Display name* field on the server, however, the intent of this procedure is to recommend verifying the true name using `snmpwalk` as opposed to relying on the service manager GUI.

Examples

Monitoring the service running state of the *Task Scheduler* on an English local Microsoft Windows® Server requires at minimum the following entry in the `poller-configuration.xml`.

```
<service name="Windows-Task-Scheduler" interval="300000" user-defined="false" status=
"on">
    <parameter key="service-name" value="Task Scheduler"/>
</service>

<monitor service="Windows-Task-Scheduler" class-name=
"org.opennms.netmgt.poller.monitors.Win32ServiceMonitor"/>
```

3.6.45. WsManMonitor

This monitor can be used to issue a WS-Man *Get* command and validate the results using a SPEL expression.

Monitor facts

Class Name	`org.opennms.netmgt.poller.monitors.WsManMonitor`
Remote Enabled	false

Configuration and Usage

Table 63. Monitor specific parameters for the WsManMonitor

Parameter	Description	Required	Default value
`resource-uri`	Resource URI	required	`-`
`rule`	SPEL expression applied against the result of the *Get*	required	`-`
`selector.`	Used to filter the result set. All selectors must prefixed with `selector.`	optional	`(None)`

Examples

The following monitor will issue a *Get* against the configured resource and verify that the correct service tag is returned:

```
<service name="WsMan-ServiceTag-Check" interval="300000" user-defined="false" status=
"on">
  <parameter key="resource-uri" value="http://schemas.dell.com/wbem/wscim/1/cim-
schema/2/root/dcim/DCIM_ComputerSystem"/>
  <parameter key="selector.CreationClassName" value="DCIM_ComputerSystem"/>
  <parameter key="selector.Name" value="srv:system"/>
  <parameter key="rule" value="#IdentifyingDescriptions matches '.*ServiceTag' and
#OtherIdentifyingInfo matches 'C7BBBP1'"/>
</service>

<monitor service="WsMan-ServiceTag-Check" class-name=
"org.opennms.netmgt.poller.monitors.WsManMonitor/>
```

3.6.46. XmpMonitor

The XMP monitor tests for *XMP service/agent* availability by establishing an *XMP* session and querying the target agent's *sysObjectID* variable contained in the *Core MIB*. The service is considered available when the session attempt succeeds and the agent returns its *sysObjectID* without error.

Monitor facts

Class Name	`org.opennms.netmgt.poller.monitors.XmpMonitor`
Remote Enabled	false

Configuration and Usage

These parameters can be set in the *XMP* service entry in *collectd-configuration.xml* and will override settings from *xmp-config.xml*. Also, don't forget to add an entry in *response-graph.properties* so that response values will be graphed.

Table 64. Monitor specific parameters for the XmpMonitor

Parameter	Description	Required	Default value
`timeout`	Time in milliseconds to wait for a successful session.	optional	`5000`
`authenUser`	The authenUser parameter for use with the XMP session.	optional	`xmpUser`
`port`	TCP port to connect to for XMP session establishment	optional	`5270`
`mib`	Name of MIB to query	optional	`core`
`object`	Name of MIB object to query	optional	`sysObjectID`

Examples

Adding entry in collectd-configuration.xml

```
<service name="XMP" interval="300000" user-defined="false" status="on">
  <parameter key="timeout" value="3000"/>
  <parameter key="rrd-repository" value="/opt/opennms/share/rrd/response"/>
  <parameter key="rrd-base-name" value="xmp"/>
  <parameter key="ds-name" value="xmp"/>
</service>
<monitor service="XMP" class-name="org.opennms.netmgt.poller.monitors.XmpMonitor"/>
```

Add entry in response-graph.properties

```
reports=icmp, \
xmp, \ . . . .

report.xmp.name=XMP
report.xmp.columns=xmp
report.xmp.type=responseTime
report.xmp.command=--title="XMP Response Time" \
 --vertical-label="Seconds" \
 DEF:rtMills={rrd1}:xmp:AVERAGE \
 DEF:minRtMills={rrd1}:xmp:MIN \
 DEF:maxRtMills={rrd1}:xmp:MAX \
 CDEF:rt=rtMills,1000,/ \
 CDEF:minRt=minRtMills,1000,/ \
 CDEF:maxRt=maxRtMills,1000,/ \
 LINE1:rt#0000ff:"Response Time" \
 GPRINT:rt:AVERAGE:" Avg  \\: %8.2lf %s" \
 GPRINT:rt:MIN:"Min  \\: %8.2lf %s" \
 GPRINT:rt:MAX:"Max  \\: %8.2lf %s\\n"
```

Chapter 4. Performance Management

4.1. Collectors

4.1.1. WS-Management

Web Services-Management (WS-Management) is a DMTF open standard defining a SOAP-based protocol for the management of servers, devices, applications and various Web services. Windows Remote Management (WinRM) is the Microsoft implementation of WS-Management Protocol. OpenNMS Horizon currently provides support for detecting, polling and collecting metrics from WS-Man endpoints.

Setup

Before setting up OpenNMS Horizon to communicate with a WS-Management agent, you should confirm that it is properly configured and reachable from the OpenNMS Horizon system. If you need help enabling the WS-Management agent, consult the documentation from the manufacturer. Here are some link resources that could help:

- Installation and Configuration for Windows Remote Management
- Troubleshooting WinRM connection and authentication

We suggest using the Openwsman command line client for validating authentication and connectivity. Packages are available for most distributions under `wsmancli`.

For example:

```
wsman identify -h localhost -P 5985 -u wsman -p secret
```

Once validated, add the agent specific details to the OpenNMS Horizon configuration, defined in the next section.

Troubleshooting and Commands

For troubleshooting there is a set of commands you can use in *Powershell* verified on *Microsoft Windows Server 2012*.

Enable WinRM in PowerShell

```
Enable-PSRemoting
```

Setup Firewall for WinRM over HTTP

```
netsh advfirewall firewall add rule name="WinRM-HTTP" dir=in localport=5985
protocol=TCP action=allow
```

Setup Firewall for WinRM over HTTPS

```
netsh advfirewall firewall add rule name="WinRM-HTTPS" dir=in localport=5986
protocol=TCP action=allow
```

Test WinRM on local Windows Server

```
winrm id
```

Show WinRM configuration on Windows Server

```
winrm get winrm/config
```

Show listener for configuration on Windows Server

```
winrm e winrm/config/listener
```

Test connectivity from a Linux system

```
nc -z -w1 <windows-server-ip-or-host> 5985;echo $?
```

Use BasicAuthentication just with *WinRM* over *HTTPS* with verifiable certificates in production environment.

Enable BasicAuthentication

```
winrm set winrm/config/client/auth '@{Basic="true"}'
winrm set winrm/config/service/auth '@{Basic="true"}'
winrm set winrm/config/service '@{AllowUnencrypted="true"}'
```

Agent Configuration

The agent specific configuration details are maintained in `etc/wsman-config.xml`. This file has a similar structure as `etc/snmp-config.xml`, which the reader may already be familiar with.

This file is consulted when a connection to a WS-Man Agent is made. If the IP address of the agent is matched by the `range`, `specific` or `ip-match` elements of a definition, then the attributes on that definition are used to connect to the agent. Otherwise, the attributes on the outer `wsman-config` definition are used.

This `etc/wsman-config.xml` files is automatically reloaded when modified.

Here is an example with several definitions:

```
<?xml version="1.0"?>
<wsman-config retry="3" timeout="1500" ssl="true" strict-ssl="false" path="/wsman">
    <definition ssl="true" strict-ssl="false" path="/wsman" username="root" password=
"calvin" product-vendor="Dell" product-version="iDRAC 6">
        <range begin="192.168.1.1" end="192.168.1.10"/>
    </definition>
    <definition ssl="false" port="5985" path="/wsman" username="Administrator"
password="P@ssword">
        <ip-match>172.23.1-4.1-255</ip-match>
        <specific>172.23.1.105</specific>
    </definition>
</wsman-config>
```

Table 65. Collector configuration attributes

Attribute	Description	Default
`timeout`	*HTTP* Connection and response timeout in milliseconds.	*HTTP* client default
`retry`	Number of retries on connection failure.	`0`
`username`	Username for basic authentication.	*none*
`password`	Password used for basic authentication.	*none*
`port`	*HTTP/S* port	*Default for protocol*
`max-elements`	Maximum number of elements to retrieve in a single request.	*no limit*
`ssl`	Enable *SSL*	`False`
`strict-ssl`	Enforce *SSL* certificate verification.	`True`
`path`	Path in the URL to the WS-Management service.	`/`
`product-vendor`	Used to overwrite the detected product vendor.	*none*
`product-version`	Used to overwrite the detected product version.	*none*
`gss-auth`	Enables GSS authentication. When enabled a reverse lookup is performed on the target IP address in order to determine the canonical host name.	`False`

If you try to connect against *Microsoft Windows Server* make sure to set specific ports for *WinRM* connections. By default *Microsoft Windows Server* uses port `TCP/5985` for plain text and port `TCP/5986` for *SSL* connections.

Collector

Configuration for the WS-Management collector is stored in `etc/wsman-datacollection-config.xml` and `etc/wsman-datacollection.d/*.xml`.

The contents of these files are automatically merged and reloaded when changed. The `default` WS-Management collection looks as follows:

```
<?xml version="1.0"?>
<wsman-datacollection-config rrd-repository="${install.share.dir}/rrd/snmp/">
    <collection name="default">
        <rrd step="300">
            <rra>RRA:AVERAGE:0.5:1:2016</rra>
            <rra>RRA:AVERAGE:0.5:12:1488</rra>
            <rra>RRA:AVERAGE:0.5:288:366</rra>
            <rra>RRA:MAX:0.5:288:366</rra>
            <rra>RRA:MIN:0.5:288:366</rra>
        </rrd>

        <!--
             Include all of the available system definitions
         -->
        <include-all-system-definitions/>
    </collection>
</wsman-datacollection-config>
```

The magic happens with the `<include-all-system-definitions/>` element which automatically includes all of the system definitions into the collection group.

If required, you can include a specific system-definition with `<include-system-definition>sys-def-name</include-system-definition>`.

System definitions and related groups can be defined in the root `etc/wsman-datacollection-config.xml` file, but it is preferred that be added to a device specific configuration files in `etc/wsman-datacollection-config.d/*.xml`.

Avoid modifying any of the distribution configuration files and create new ones to store you specific details instead.

Here is an example configuration file for a *Dell iDRAC*:

```
<?xml version="1.0"?>
<wsman-datacollection-config>
    <group name="drac-system"
           resource-uri="http://schemas.dell.com/wbem/wscim/1/cim-
schema/2/root/dcim/DCIM_ComputerSystem"
           resource-type="node">
        <attrib name="OtherIdentifyingInfo" index-of="#IdentifyingDescriptions matches
'.*ServiceTag'" alias="serviceTag" type="String"/>
    </group>

    <group name="drac-power-supply"
           resource-uri="http://schemas.dmtf.org/wbem/wscim/1/*"
           dialect="http://schemas.microsoft.com/wbem/wsman/1/WQL"
           filter="select
InputVoltage,InstanceID,PrimaryStatus,SerialNumber,TotalOutputPower from
DCIM_PowerSupplyView where DetailedState != 'Absent'"
           resource-type="dracPowerSupplyIndex">
        <attrib name="InputVoltage" alias="inputVoltage" type="Gauge"/>
        <attrib name="InstanceID" alias="instanceId" type="String"/>
        <attrib name="PrimaryStatus" alias="primaryStatus" type="Gauge"/>
        <attrib name="SerialNumber" alias="serialNumber" type="String"/>
        <attrib name="TotalOutputPower" alias="totalOutputPower" type="Gauge"/>
    </group>

    <system-definition name="Dell iDRAC (All Version)">
        <rule>#productVendor matches '^Dell.*' and #productVersion matches
'.*iDRAC.*'</rule>
        <include-group>drac-system</include-group>
        <include-group>drac-power-supply</include-group>
    </system-definition>
</wsman-datacollection-config>
```

System Definitions

Rules in the system definition are written using SpEL expressions.

The expression has access to the following variables in it`s evaluation context:

Name	Type
(root)	*org.opennms.netmgt.model.OnmsNode*
agent	*org.opennms.netmgt.collection.api.CollectionAgent*
productVendor	*java.lang.String*
productVersion	*java.lang.String*

If a particular agent is matched by any of the rules, then the collector will attempt to collect the referenced groups from the agent.

Group Definitions

Groups are retrieved by issuing an Enumerate command against a particular `Resource URI` and parsing the results. The Enumerate commands can include an optional `filter` in order to filter the records and attributes that are returned.

When configuring a filter, you must also specify the dialect.

The resource type used by the group must of be of type `node` or a generic resource type. Interface level resources are not supported.

When using a generic resource type, the `IndexStorageStrategy` cannot be used since records have no implicit index. Instead, you must use an alternative such as the `SiblingColumnStorageStrategy`.

If a record includes a multi-valued key, you can collect the value at a specific index with an `index-of` expression. This is best demonstrated with an example. Let`s assume we wanted to collect the `ServiceTag` from the following record:

```
<IdentifyingDescriptions>CIM:GUID</IdentifyingDescriptions>
<IdentifyingDescriptions>CIM:Tag</IdentifyingDescriptions>
<IdentifyingDescriptions>DCIM:ServiceTag</IdentifyingDescriptions>
<OtherIdentifyingInfo>45454C4C-3700-104A-8052-C3C01BB25031</OtherIdentifyingInfo>
<OtherIdentifyingInfo>mainsystemchassis</OtherIdentifyingInfo>
<OtherIdentifyingInfo>C8BBBP1</OtherIdentifyingInfo>
```

Specifying, the attribute name `OtherIdentifyingInfo` would not be sufficient, since there are multiple values for that key. Instead, we want to retrieve the value for the `OtherIdentifyingInfo` key at the same index where `IdentifyingDescriptions` is set to `DCIM:ServiceTag`.

This can be achieved using the following attribute definition:

```
<attrib name="OtherIdentifyingInfo" index-of="#IdentifyingDescriptions matches
'.*ServiceTag'" alias="serviceTag" type="String"/>
```

Detector

The WS-Management detector attempts to connect to the agent defined in `wsman-config.xml` and issues an Identify command. If a valid response is received, the product vendor and product version are stored in the `vendor` and `modelNumber` fields of the associated node`s assets table.

For example, a Windows Server 2008 machine returns:

Product Vendor	Microsoft Corporation
Product Version	OS: 6.1.7601 SP: 1.0 Stack: 2.0

If these assets field are being used for another purpose, this behavior can be disabled by settings the `updateAssets` parameters to `false` in the detector configuration of the appropriate foreign

source.

Some agents may respond to the Identify command with generic identities such as `Openwsman 2.0.0`. These values can be overridden by specifying the `product-vendor` and `product-version` attributes in `wsman-config.xml`.

Example detector configuration:

```
<detector name="WS-Man" class=
"org.opennms.netmgt.provision.detector.wsman.WsManDetector">
    <parameter key="updateAssets" value="true"/>
</detector>
```

The response is logged as *DEBUG* information in `provisiond.log` and looks like the following:

```
ID: 3
Response-Code: 200
309Encoding: UTF-8
Content-Type: application/soap+xml;charset=UTF-8
Headers: {Content-Length=[787], content-type=[application/soap+xml;charset=UTF-8],
Date=[Mon, 08 Feb 2016 14:21:20 GMT], Server=[Microsoft-HTTPAPI/2.0]}
Payload:
<s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope" xml:lang="en-US">
  <s:Header/>
  <s:Body>
    <wsmid:IdentifyResponse xmlns:wsmid=
"http://schemas.dmtf.org/wbem/wsman/identity/1/wsmanidentity.xsd">
    <wsmid:ProtocolVersion>
http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd</wsmid:ProtocolVersion>
    <wsmid:ProductVendor>Microsoft Corporation</wsmid:ProductVendor> ①
    <wsmid:ProductVersion>OS: 6.2.9200 SP: 0.0 Stack: 3.0</wsmid:ProductVersion> ②
    <wsmid:SecurityProfiles>

<wsmid:SecurityProfileName>http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/http/
basic</wsmid:SecurityProfileName>

<wsmid:SecurityProfileName>http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/http/
spnego-kerberos</wsmid:SecurityProfileName>
    </wsmid:SecurityProfiles>
    </wsmid:IdentifyResponse>
  </s:Body>
</s:Envelope>
```

① `ProductVendor`: Stored to the asset field `vendor`

② `ProductVersion`: Stored in the asset field `modelNumber`

The information of the asset fields are used in the *System Definition Rule* to decide which performance metrics will be gathered from *Collectd*.

Chapter 5. Events

Events are central to the operation of the OpenNMS Horizon platform, so it's critical to have a firm grasp of this topic.

Whenever something in OpenNMS Horizon appears to work by magic, it's probably events working behind the curtain.

5.1. Anatomy of an Event

Events are structured historical records of things that happen in OpenNMS Horizon and the nodes, interfaces, and services it manages. Every event has a number of fixed **fields** and zero or more **parameters**.

Mandatory Fields

UEI (Universal Event Identifier)

A string uniquely identifying the event's type. UEIs are typically formatted in the style of a URI, but the only requirement is that they start with the string `uei.`.

Event Label

A short, static label summarizing the gist of all instances of this event.

Description

A long-form description describing all instances of this event.

Log Message

A long-form log message describing this event, optionally including expansions of fields and parameters so that the value is tailored to the event at hand.

Severity

A severity for this event type. Possible values range from `Cleared` to `Critical`.

Event ID

A numeric identifier used to look up a specific event in the OpenNMS Horizon system.

Notable Optional Fields

Operator Instruction

A set of instructions for an operator to respond appropriately to an event of this type.

Alarm Data

If this field is provided for an event, OpenNMS Horizon will create, update, or clear **alarms** for events of that type according to the alarm-data specifics. For more about alarms and how they relate to events, see [alarms-introduction].

5.2. Sources of Events

Events may originate within OpenNMS Horizon itself or from outside.

Internally-generated events can be the result of the platform's monitoring and management functions (*e.g.* a monitored node becoming totally unavailable results in an event with the UEI `uei.opennms.org/nodes/nodeDown`) or they may act as inputs or outputs of housekeeping processes.

The following subsections summarize the mechanisms by which externally-created events can arrive.

5.2.1. SNMP Traps

If SNMP-capable devices in the network are configured to send **traps** to OpenNMS Horizon, these traps are transformed into events according to pre-configured rules. The `Trapd` service daemon, which enables OpenNMS Horizon to receive SNMP traps, is enabled by default.

Disabling the `Trapd` service daemon will render OpenNMS Horizon **incapable** of receiving SNMP traps.

Event definitions are included with OpenNMS Horizon for traps from many vendors' equipment.

5.2.2. Syslog Messages

Syslog messages sent over the network to OpenNMS Horizon can be transformed into events according to pre-configured rules.

The `Syslogd` service daemon, which enables OpenNMS Horizon to receive syslog messages over the network, must be enabled for this functionality to work. This service daemon is **disabled** by default.

5.2.3. TL1 Autonomous Messages

Autonomous messages can be retrieved from certain TL1-enabled equipment and transformed into events.

The `Tl1d` service daemon, which enables OpenNMS Horizon to receive TL1 autonomous messages, must be enabled for this functionality to work. This service daemon is **disabled** by default.

5.2.4. XML-TCP

Any application or script can create custom events in OpenNMS Horizon by sending properly-formatted XML data over a TCP socket.

5.2.5. ReST

Posting an event in XML format to the appropriate endpoint in the OpenNMS Horizon ReST API will

cause the creation of a corresponding event, just as with the XML-TCP interface.

5.3. The Event Bus

At the heart of OpenNMS Horizon lies an **event bus**. Any OpenNMS Horizon component can *publish* events to the bus, and any component can *subscribe* to receive events of interest that have been published on the bus. This publish-subscribe model enables components to use events as a mechanism to send messages to each other. For example, the provisioning subsystem of OpenNMS Horizon publishes a *node-added* event whenever a new node is added to the system. Other subsystems with an interest in new nodes subscribe to the *node-added* event and automatically receive these events, so they know to start monitoring and managing the new node if their configuration dictates. The publisher and subscriber components do not need to have any knowledge of each other, allowing for a clean division of labor and lessening the programming burden to add entirely new OpenNMS Horizon subsystems or modify the behavior of existing ones.

5.4. Forwarding Events to Elasticsearch 1.x

OpenNMS can be configured to forward all *Events* and *Alarms* to Elasticsearch 1.x for indexing, long time archiving, plotting with *Grafana* and browsing with *Kibana*.

Elasticsearch is not intended as a replacement for *PostgreSQL* which is still a required component to run *OpenNMS*.

First check that your *OpenNMS* installation supports this feature. If it does there should be a `${OPENNMS_HOME}/etc/org.opennms.features.elasticsearch.eventforwarder.cfg` file.

Now open the file, review its content and make sure to apply the correct settings depending on your environment.

The following table describes all settings and possible values.

Parameter	Default	Description
`elasticsearchCluster`	opennms	The name of the *Elasticsearch* cluster as specified in the *Elasticsearch* configuration file (required).
`elasticsearchIp`	localhost	the *TransportClient* remote host ip to use. Has the same meaning as the ip options of the camel-elasticsearch component
`logEventDescription`	false	Whether to forward the event description to *Elasticsearch*. The reason it is off by default is that it is usually some standard, generic, repetitive and possibility long text which will grow the index without adding useful information.
`cache_max_ttl`	0	The number of minutes the node information is kept in the cache. Set to `0` to disable (which is the default and is generally safe because the cache knows when to refresh itself, by intercepting nodeUpdated and similar events)

Parameter	Default	Description
`cache_max_size`	10000	The number of node information entries to be kept in the cache before eviction start. Set to `0` to disable.

The first two (`elasticsearchCluster` and `elasticsearchIp`) settings are the most likely to require changing. If unsure do not change the remaining three.

Once you are sure everything is correctly configured you can activate the *Elasticsearch* forwarder by log into the *OSGi* console and install the feature.

OSGi login and installation of the Elasticsearch forwarder

```
ssh admin@localhost -p 8101
features:install opennms-elasticsearch-event-forwarder
```

You can check the routes status with the `camel:*` commands and/or inspect the log with `log:tail` for any obvious errors. The feature has a trace level logging that can be used to trace operations.

documentation on using the *OSGi* console embedded in *OpenNMS* and the relative camel commands.

If all goes well events and alarms will be pushed in realtime into *Elasticsearch*. You should now be able to view the events and graph them with Kibana.

If you have never used *Kibana* before you should probably start with *Kibana 3* which is simpler. *Kibana 4* is more powerful, but harder to get started with.

5.4.1. A basic Elasticsearch configuration

This section describes to get a minimal working configuration with *OpenNMS* and *Elasticsearch*. Install *Elasticsearch* on the same host as *OpenNMS* and edit the `elasticsearch.yml` as follows:

Example configuration for Elasticsearch

```
cluster.name: opennms
network.host: 127.0.0.1
discovery.zen.ping.multicast.enabled: false
discovery.zen.ping.unicast.hosts: ["127.0.0.1"]
```

Running *OpenNMS* and *Elasticsearch* on the same host is not recommended for production or busy environments.

5.4.2. Troubleshooting

If events are not reaching *Elasticsearch* check if *OpenNMS* is correctly configured, in particular review the `elasticsearchCluster` and `elasticsearchIp` parameters.

If those appear to be correct verify that *OpenNMS* can communicate with *Elasticsearch* over port 9300.

Review the OSGi log with `log:tail` or the `camel:*` commands.

Chapter 6. Provisioning

6.1. Introduction

The introduction of OpenNMS version 1.8 empowers enterprises and services providers like never before with a new service daemon for maintaining the managed entity inventory in OpenNMS. This new daemon, *Provisiond*, unifies all previous entity control mechanisms available in 1.6 (*Capsd* and the *Importer*), into a new and improved, massively parallel, policy based provisioning system. System integrators should note, *Provisiond* comes complete with a *RESTFul Web Service API* for easy integration with external systems such as CRM or external inventory systems as well as an adapter API for interfacing with other management systems such as configuration management.

OpenNMS 1.0, introduced almost a decade ago now, provided a capabilities scanning daemon, Capsd, as the mechanism for provisioning managed entities. *Capsd*, deprecated with the release of 1.8.0, provided a rich automatic provisioning mechanism that simply required an IP address to seed its algorithm for creating and maintaining the managed entities (nodes, interfaces, and IP based services). Version 1.2 added and *XML-RPC API* as a more controlled (directed) strategy for provisioning services that was mainly used by non telco based service providers (i.e. managed hosting companies). Version 1.6 followed this up with yet another and more advanced mechanism called the *Importer service daemon*. The *Importer* provided large service providers with the ability to strictly control the OpenNMS entity provisioning with an XML based API for completely defining and controlling the entities where no discovery and service scanning scanning was feasible.

The Importer service improved OpenNMS' scalability for maintaining managed entity databases by an order of magnitude. This daemon, while very simple in concept and yet extremely powerful and flexible provisioning improvement, has blazed the trail for Provisiond. The Importer service has been in production for 3 years in service provider networks maintaining entity counts of more than 50,000 node level entities on a single instances of OpenNMS. It is a rock solid provisioning tool.

Provisiond begins a new era of managed entity provisioning in OpenNMS.

6.2. Concepts

Provisioning is a term that is familiar to service providers (a.k.a. operators, a.k.a. telephone companies) and OSS systems but not so much in the non OSS enterprises.

Provisiond receives "requests" for adding managed entities via 2 basic mechanisms, the OpenNMS Horizon traditional "New Suspect" event, typically via the *Discovery daemon*, and the import requisition (XML definition of node entities) typically via the Provisioning Groups UI. If you are familiar with all previous releases of OpenNMS, you will recognize the *New Suspect Event* based *Discovery* to be what was previously the *Capsd* component of the auto discovery behavior. You will also recognize the import requisition to be of the *Model Importer* component of OpenNMS. *Provisiond* now unifies these two separate components into a massively parallel advanced policy based provisioning service.

6.2.1. Terminology

The following terms are used with respect to the OpenNMS Horizon provisioning system and are essential for understanding the material presented in this guide.

Entity

Entities are managed objects in OpenNMS Horizon such as Nodes, IP interfaces, SNMP Interfaces, and Services.

Foreign Source and Foreign ID

The *Importer* service from 1.6 introduced the idea of foreign sources and foreign IDs. The *Foreign Source* uniquely identifies a provisioning source and is still a basic attribute of importing node entities into OpenNMS Horizon. The concept is to provide an external (foreign) system with a way to uniquely identify itself and any node entities that it is requesting (via a requisition) to be provisioned into OpenNMS Horizon.

The *Foreign ID* is the unique node ID maintained in foreign system and the foreign source uniquely identifies the external system in OpenNMS Horizon.

OpenNMS Horizon uses the combination of the foreign source and foreign ID become the unique foreign key when synchronizing the set of nodes from each source with the nodes in the OpenNMS Horizon DB. This way the foreign system doesn't have to keep track of the OpenNMS Horizon node IDs that are assigned when a node is first created. This is how *Provisiond* can decided if a node entity from an import requisition is new, has been changed, or needs to be deleted.

Foreign Source Definition

Additionally, the foreign source has been extended to also contain specifications for how entities should be discovered and managed on the nodes from each foreign source. The name of the foreign source has become pervasive within the provisioning system and is used to simply some of the complexities by weaving this name into:

- the name of the provisioning group in the Web-UI
- the name of the file containing the persisted requisition (as well as the pending requisition if it is in this state)
- the foreign-source attribute value inside the requisition (obviously, but, this is pointed out to indicate that the file name doesn't necessarily have to equal the value of this attribute but is highly recommended as an OpenNMS Horizon best practice)
- the building attribute of the node defined in the requisition (this value is called "site" in the Web-UI and is assigned to the building column of the node's asset record by Provisiond and is the default value used in the Site Status View feature)

Import Requisition

Import requisition is the terminology OpenNMS Horizon uses to represent the set of nodes, specified in XML, to be provisioned from a foreign source into OpenNMS Horizon. The requisition schema (XSD) can be found at the following location. http://xmlns.opennms.org/xsd/config/model-

import

Auto Discovery

Auto discovery is the term used by OpenNMS Horizon to characterize the automatic provisioning of nodes entities. Currently, OpenNMS Horizon uses an ICMP ping sweep to find IP address on the network. For the IPs that respond and that are not currently in the DB, OpenNMS Horizon generates a new suspect event. When this event is received by Provisiond, it creates a node and it begins a node scan based on the default foreign source definition.

Directed Discovery

Provisiond takes over for the Model Importer found in version 1.6 which implemented a unique, first of its kind, controlled mechanism for specifying managed entities directly into OpenNMS Horizon from one or more data sources. These data sources often were in the form of an in-housed developed inventory or stand-alone provisioning system or even a set of element management systems. Using this mechanism, OpenNMS Horizon is directed to add, update, or delete a node entity exactly as defined by the external source. No discovery process is used for finding more interfaces or services.

Enhanced Directed Discovery

Directed discovery is enhanced with the capability to scan nodes that have been directed nodes for entities (interfaces.

Policy Based Discovery

The phrase, Policy based Directed Discovery, is a term that represents the latest step in OpenNMS Horizon provisioning evolution and best describes the new provisioning architecture now in OpenNMS Horizon for maintaining its inventory of managed entities. This term describes the control that is given over the Provisioning system to OpenNMS Horizon users for managing the behavior of the NMS with respect to the new entities that are being discovered. Current behaviors include persistence, data collection, service monitoring, and categorization policies.

6.2.2. Addressing Scalability

The explosive growth and density of the IT systems being deployed today to support not traditional IP services is impacting management systems like never before and is demanding from them tremendous amounts of scalability. The scalability of a management system is defined by its capacity for maintaining large numbers of managing entities coupled with its efficiency of managing the entities.

Today, It is not uncommon for OpenNMS Horizon deployments to find node entities with tens of thousands of physical interfaces being reported by SNMP agents due to virtualization (virtual hosts, interfaces, as well as networks). An NMS must be capable of using the full capacity every resource of its computing platform (hardware and OS) as effectively as possible in order to manage these environments. The days of writing scripts or single threaded applications will just no longer be able to do the work required an NMS when dealing with the scalability challenges facing systems and systems administrators working in this domain.

Parallelization and Non-Blocking I/O

Squeezing out every ounce of power from a management system's platform (hardware and OS) is absolutely required to complete all the work of a fully functional NMS such as OpenNMS Horizon. Fortunately, the hardware and CPU architecture of a modern computing platform provides multiple CPUs with multiple cores having instruction sets that include support for atomic operations. While these very powerful resources are being provided by commodity systems, it makes the complexity of developing applications to use them vs. not using them, orders of magnitude more complex. However, because of scalability demands of our complex IT environments, multi-threaded NMS applications are now essential and this has fully exposed the complex issues of concurrency in software development.

OpenNMS Horizon has stepped up to this challenge with its new concurrency strategy. This strategy is based on a technique that combines the efficiency of parallel (asynchronous) operations (traditionally used by most effectively by single threaded applications) with the power of a fully current, non-blocking, multi-threaded design. The non-blocking component of this new concurrency strategy added greater complexity but OpenNMS Horizon gained orders of magnitude in increased scalability.

Java Runtimes, based on the Sun JVM, have provided implementations for processor based atomic operations and is the basis for OpenNMS Horizon' non-blocking concurrency algorithms.

Provisioning Policies

Just because you can, doesn't mean you should! Because the massively parallel operations being created for *Provisiond* allows tremendous numbers of nodes, interfaces, and services to be very rapidly discovered and persisted, doesn't mean it should. A *policy API* was created for *Provisiond* that allows implementations to be developed that can be applied to control the behavior of *Provisiond*. The 1.8 release includes a set of flexible provisioning policies that control the persistence of entities and their attributes constrain monitoring behavior.

When nodes are imported or re-scanned, there is, potentially, a set of zero or more provisioning policies that are applied. The policies are defined in the foreign source's definition. The policies for an auto-discovered node or nodes from provisioning groups that don't have a foreign source definition, are the policies defined in the default foreign source definition.

The Default Foreign Source Definition

Contained in the libraries of the Provisioning service is the "template" or default foreign source. The template stored in the library is used until the OpenNMS Horizon admin user alters the default from the *Provisioning Groups* WebUI. Upon edit, this template is exported to the OpenNMS Horizon `etc/` directory with the file name: `default-foreign-source.xml.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<foreign-source date-stamp="2009-10-16T18:04:12.844-05:00"
                name="default"
                xmlns=
"http://xmlns.opennms.org/[http://xmlns.opennms.org/xsd/config/foreign-source">
    <scan-interval>1d</scan-interval>
    <detectors>
      <detector class="org.opennms.netmgt.provision.detector.datagram.DnsDetector"
name="DNS"/>
      <detector class="org.opennms.netmgt.provision.detector.simple.FtpDetector" name
="FTP"/>
      <detector class="org.opennms.netmgt.provision.detector.simple.HttpDetector"
name="HTTP"/>
      <detector class="org.opennms.netmgt.provision.detector.simple.HttpsDetector"
name="HTTPS"/>
      <detector class="org.opennms.netmgt.provision.detector.icmp.IcmpDetector" name=
"ICMP"/>
      <detector class="org.opennms.netmgt.provision.detector.simple.ImapDetector"
name="IMAP"/>
      <detector class="org.opennms.netmgt.provision.detector.simple.LdapDetector"
name="LDAP"/>
      <detector class="org.opennms.netmgt.provision.detector.simple.NrpeDetector"
name="NRPE"/>
      <detector class="org.opennms.netmgt.provision.detector.simple.Pop3Detector"
name="POP3"/>
      <detector class="
org.opennms.netmgt.provision.detector.radius.RadiusAuthDetector" name="Radius"/>
      <detector class="org.opennms.netmgt.provision.detector.simple.SmtpDetector"
name="SMTP"/>
      <detector class="org.opennms.netmgt.provision.detector.snmp.SnmpDetector" name=
"SNMP"/>
      <detector class="org.opennms.netmgt.provision.detector.ssh.SshDetector" name=
"SSH"/>
  </detectors>
  <policies/>
</foreign-source>
```

Default Foreign Source

6.3. Getting Started

An NMS is of no use until it is setup for monitoring and entities are added to the system. OpenNMS Horizon installs with a base configuration with a configuration that is sufficient get service level monitoring and performance management quickly up and running. As soon as managed entities are provisioned, the base configuration will automatically begin monitoring and reporting.

Generally speaking, there are two methods of provisioning in OpenNMS Horizon: *Auto Discovery* and *Directed Discovery*. We'll start with *Auto Discovery*, but first, we should quickly review the configuration of SNMP so that newly discovered devices can be immediately scanned for entities as

well as have reporting and thresholding available.

6.3.1. Provisioning the SNMP Configuration

OpenNMS Horizon requires that the SNMP configuration to be properly setup for your network in order to properly understand Network and Node topology as well as to automatically enabled performance data collection. Network topology is updated as nodes (a.k.a. devices or hosts) are provisioned. Navigate to the *Admin/Configure SNMP Community Names* as shown below.

Provisiond includes an option to add community information in the *Single Node* provisioning interface. This, is equivalent of entering a single IP address in the screen with the convenience of setting the community string at the same time a node is provisioned. See the *Quick Node Add* feature below for more details about this capability.

This screen sets up SNMP within OpenNMS Horizon for agents listening on IP addresses 10.1.1.1 through 10.254.254.254. These settings are optimized into the `snmp-configuration.xml` file. Optimization means that the minimal configuration possible will be written. Any IP addresses already configured that are eclipsed by this range will be removed. Here is the resulting configuration.

Sample snmp-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<snmp-config
xmlns="http://xmlns.opennms.org/xsd/config/snmp[http://xmlns.opennms.org/xsd/config/sn
mp]"
port="161" retry="3" timeout="800" read-community="public"

version="v1" max-vars-per-pdu="10">

<definition retry="1" timeout="2000"

read-community="public" version="v2c">

<specific>10.12.23.32</specific>

</definition>

</snmp-config>
```

However, If an IP address is then configured that is within the range, the range will be split into two separate ranges and a specific entry is added. For example, if a configuration was added through the same UI for the IP: 10.12.23.32 having the community name `public`, then the resulting configuration will be:

```
<?xml version="1.0" encoding="UTF-8"?>
<snmp-config xmlns="http://xmlns.opennms.org/xsd/config/snmp"
             port="161"
             retry="3"
             timeout="800"
             read-community="public"
             version="v1"
             max-vars-per-pdu="10">

    <definition retry="1" timeout="2000" read-community="YrusoNoz" version="v2c">
        <range begin="10.1.1.1" end="10.12.23.31"/>
        <range begin="10.12.23.33" end="10.254.254.254"/>
    </definition>

    <definition retry="1" timeout="2000" read-community="public" version="v2c">
        <specific>10.12.23.32</specific>
    </definition>
</snmp-config>
```

the bold IP addresses show where the range was split and the specific with community name "public" was added.

Now, with SNMP configuration provisioned for our 10 network, we are ready to begin adding nodes. Our first example will be to automatically discovery and add all managed entities (nodes, IP interfaces, SNMP Interfaces, and Monitored IP based Services). We will then give an example of how to be more *directed* and deliberate about your discovery by using *Provisioning Groups*.

Automatically discovered entities are analyzed, persisted to the relational data store, and then managed based on the policies defined in the default foreign source definition. This is very similar to the way that entities were handled by Capsd by with finer grained sense of control.

6.3.2. Automatic Discovery

Currently in OpenNMS Horizon, the ICMP is used to automatically provision node entities into OpenNMS Horizon. This functionality has been in OpenNMS since is 1.0 release, however, in 1.8, a few of the use cases have been updated with *Provisiond's* replacement of *Capsd*.

Separation of Concerns

Version 1.8 *Provisiond* separates what was called *Capsd* scanning in to 3 distinct phases: entity scanning, service detection, and node merging. These phases are now managed separately by Provisiond. Immediately following the import of a node entity, tasks are created for scanning a node to discover the node entity's interfaces (SNMP and IP). As interfaces are found, they are persisted and tasks are scheduled for service detection of each IP interface.

For auto discovered nodes, a node merging phase is scheduled; Nodes that have been directly provisioned will not be included in the node merging process. Merging will only occur when 2 automatically discovered nodes appear to be the same node.

the use case and redesign of node merging is still an outstanding issue with the 1.8.0 release

6.3.3. Enhanced Directed Discovery

This new form of provisioning first appears in OpenNMS with version 1.8 and the new Provisiond service. It combines the benefits of the Importer's strictly controlled methodology of directed provisioning (from version 1.6) with OpenNMS' robustly flexible auto discovery. *Enhanced Directed discovery* begins with an enhanced version of the same import requisition used in directed provisioning and completes with a policy influenced persistence phase that sorts though the details of all the entities and services found during the entity and service scanning phase.

If you are planning to use this form of provisioning, it important to understand the conceptual details of how *Provisiond* manages entities it is *directed* to provision. This knowledge will enable administrators and systems integrators to better plan, implement, and resolve any issues involved with this provisioning strategy.

Understanding the Process

There are 3 phases involved with directing entities to be discovered: import, node scan, and service scan. The import phase also has sub phases: marshal, audit, limited SNMP scan, and re-parent.

Marshal and Audit Phases

It is important to understand that the nodes requisitioned from each foreign source are managed as a complete set. Nodes defined in a requisition from the foreign source *CRM* and *CMDB*, for example, will be managed separately from each other even if they should contain exactly the same node definitions. To OpenNMS Horizon, these are individual entities and they are managed as a set.

Requisitions are referenced via a URL. Currently, the URL can be specified as one of the following protocols: FILE, HTTP, HTTPS, and DNS. Each protocol has a protocol handler that is used to stream the XML from a *foreign source*, i.e. http://inv.corp.org/import.cgi?customer=acme or `file:/opt/opennms/etc/imports/acme.xml`. The DNS protocol is a special handler developed for Provisioning sets of nodes as a *foreign-source* from a corporate DNS server. See DNS Protocol Handler for details.

Upon the import request (either on schedule or on demand via an Event) the requisition is marshaled into Java objects for processing. The nodes defined in the requisition represent what OpenNMS Horizon should have as the current set of managed entities from that foreign source. The audit phase determines for each node defined (or not defined) in the requisition which are to be processed as an *Add*, *Update*, or *Delete* operation during the *Import Phase*. This determination is made by comparing the set foreign IDs of each node in the requisition set with the set of foreign IDs of currently managed entities in OpenNMS Horizon.

The intersection of the IDs from each set will become the Update operations, the extra set of foreign IDs that are in the requisition become the Add operations, and the extra set of foreign IDs from the managed entities become the Delete operations. This implies that the foreign IDs from each foreign source must be unique.

Naturally, the first time an import request is processed from a foreign source there will be zero (0) node entities from the set of nodes currently being managed and each node defined in the requisition will become an Add Operation. If a requisition is processed with zero (0) node definitions, all the currently managed nodes from that foreign source will become Delete operations (all the nodes, interfaces, outages, alarms, etc. will be removed from OpenNMS Horizon).

When nodes are provisioned using the Provisioning Groups Web-UI, the requisitions are stored on the local file system and the file protocol handler is used to reference the requisition. Each Provisioning Group is a separate foreign source and unique foreign IDs are generated by the Web-UI. An MSP might use Provisioning Groups to define the set of nodes to be managed by customer name where each customer's set of nodes are maintained in a separate Provisioning Group.

Import Phase

The import phase begins when Provisiond receives a request to import a requisition from a URL. The first step in this phase is to load the requisition and marshal all the node entities defined in the requisition into Java objects.

If any syntactical or XML structural problems occur in the requisition, the entire import is abandoned and no import operations are completed.

Once the requisition is marshaled, the requisition nodes are audited against the persisted node entities. The set of requisitioned nodes are compared with a subset of persisted nodes and this subset is generated from a database query using the foreign source defined in the requisition. The audit generates one of three operations for each requisition node: *insert*, *update*, *delete* based on each requisitioned node's foreign ID. Delete operations are created for any nodes that are not in the requisition but are in the DB subset, update operations are created for requisition nodes that match a persisted node from the subset (the intersection), and insert operations are created from the remaining requisition nodes (nodes in the requisition that are not in the DB subset).

If a requisition node has an interface defined as the Primary SNMP interface, then during the update and insert operations the node will be scanned for minimal SNMP attribute information. This scan find the required node and SNMP interface details required for complete SNMP support of the node and only the IP interfaces defined in the requisition.

this not the same as Provisiond SNMP discovery scan phases: node scan and interface scan.

Node Scan Phase

Where directed discovery leaves off and enhanced directed discovery begins is that after all the operations have completed, directed discovery is finished and enhanced directed discovery takes off. The requisitioned nodes are scheduled for node scans where details about the node are discovered and interfaces that were not directly provisioned are also discovered. All physical (SNMP) and logical (IP) interfaces are discovered and persisted based on any *Provisioning Policies* that may have defined for the foreign source associated with the import requisition.

Service Scan (detection) Phase

Additionally, the new Provisiond enhanced directed discovery mechanism follows interface

discovery with service detection on each IP interface entity. This is very similar to the Capsd plugin scanning found in all former releases of OpenNMS accept that the foreign source definition is used to define what services should be detected on these interfaces found for nodes in the import requisition.

6.4. Import Handlers

6.4.1. File Handler

6.4.2. HTTP Handler

6.4.3. DNS Handler

The new Provisioning service in OpenNMS Horizon is continuously improving and adapting to the needs of the community.

One of the most recent enhancements to the system is built upon the very flexible and extensible API of referencing an import requisition's location via a URL. Most commonly, these URLs are files on the file system (i.e. `file:/opt/opennms/etc/imports/<my-provisioning-group.xml>`) as requisitions created by the Provisioning Groups UI. However, these same requisitions for adding, updating, and deleting nodes (based on the original model importer) can also come from URLs specifying the HTTP protocol: http://myinventory.server.org/nodes.cgi

Now, using Java's extensible protocol handling specification, a new protocol handler was created so that a URL can be specified for requesting a *Zone Transfer (AXFR) request* from a DNS server. The A records are recorded and used to build an import requisition. This is handy for organizations that use DNS (possibly coupled with an IP management tool) as the data base of record for nodes in the network. So, rather than ping sweeping the network or entering the nodes manually into OpenNMS Horizon Provisioning UI, nodes can be managed via 1 or more DNS servers.

The format of the URL for this new protocol handler is: dns://<host>[:port]/<zone>[/<foreign-source>/][?expression=<regex>]

DNS Import Examples:

Simple

```
dns://my-dns-server/myzone.com
```

This URL will import all A records from the host `my-dns-server` on port 53 (default port) from zone "myzone.com" and since the foreign source (a.k.a. the provisioning group) is not specified it will default to the specified zone.

Using a Regular Expression Filter

```
dns://my-dns-server/myzone.com/portland/?expression=^por-.*
```

This URL will import all nodes from the same server and zone but will only manage the nodes in

the zone matching the regular expression `^port-.*` and will and they will be assigned a unique foreign source (provisioning group) for managing these nodes as a subset of nodes from within the specified zone.

If your expression requires URL encoding (for example you need to use a `?` in the expression) it must be properly encoded.

```
dns://my-dns-server/myzone.com/portland/?expression=^por[0-9]%3F
```

DNS Setup

Currently, the DNS server requires to be setup to allow a zone transfer from the OpenNMS Horizon server. It is recommended that a secondary DNS server is running on OpenNMS Horizon and that the OpenNMS Horizon server be allowed to request a zone transfer. A quick way to test if zone transfers are working is:

```
dig -t AXFR @<dnsServer> <zone>
```

Configuration

The configuration of the Provisoning system has moved from a properties file (`model-importer.properties`) to an XML based configuration container. The configuration is now extensible to allow the definition of 0 or more import requisitions each with their own cron based schedule for automatic importing from various sources (intended for integration with external URL such as http and this new dns protocol handler.

A default configuration is provided in the OpenNMS Horizon `etc/` directory and is called: `provisiond-configuration.xml`. This default configuration has an example for scheduling an import from a DNS server running on the localhost requesting nodes from the zone, localhost and will be imported once per day at the stroke of midnight. Not very practical but is a good example.

```
<?xml version="1.0" encoding="UTF-8"?>
    <provisiond-configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.opennms.org/xsd/config/provisiond-configuration"
                              foreign-source-dir="/opt/opennms/etc/foreign-sources"
                              requistion-dir="/opt/opennms/etc/imports"
                              importThreads="8"
                              scanThreads="10"
                              rescanThreads="10"
                              writeThreads="8" >

    <!--http://www.quartz-scheduler.org/documentation/quartz-1.x/tutorials/crontrigger
        Field Name Allowed Values Allowed Special Characters
        Seconds 0-59 , - * / Minutes 0-59 , - * / Hours 0-23 , - * /
        Day-of-month1-31, - * ? / L W C Month1-12 or JAN-DEC, - * /
        Day-of-Week1-7 or SUN-SAT, - * ? / L C # Year (Opt)empty, 1970-2099, - * /
    -->

    <requisition-def import-name="localhost"
                     import-url-resource="dns://localhost/localhost">

        <cron-schedule>0 0 0 * * ? *</cron-schedule> <!-- daily, at midnight -->
    </requisition-def>
</provisiond-configuration>
```

Configuration Reload

Like many of the daemon configuration in the 1.7 branch, the configurations are reloadable without having to restart OpenNMS Horizon, using the reloadDaemonConfig uei:

```
/opt/opennms/bin/send-event.pl
uei.opennms.org/internal/reloadDaemonConfig --parm 'daemonName Provisiond'
```

This means that you don't have to restart OpenNMS Horizon every time you update the configuration.

6.5. Provisioning Examples

Here are a few practical examples of enhanced directed discovery to help with your understanding of this feature.

6.5.1. Basic Provisioning

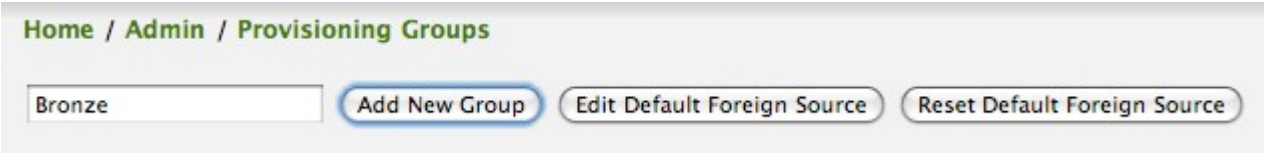
This example adds three nodes and requires no OpenNMS Horizon configuration other than specifying the node entities to be provisioned and managed in OpenNMS Horizon.

Defining the Nodes via the Web-UI

Using the Provisioning Groups Web-UI, three nodes are created given a single IP address. Navigate

to the Admin Menu and click Provisioning Groups Menu from the list of Admin options and create the group *Bronze*.

Creating a new Provisioning Group

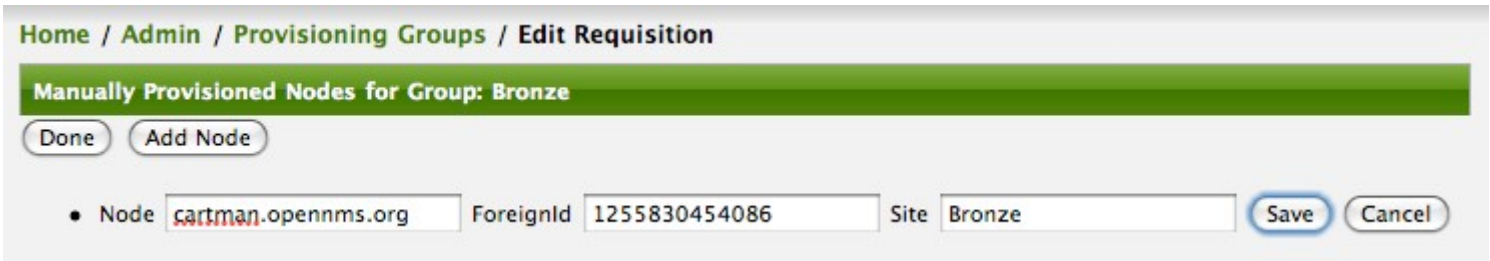


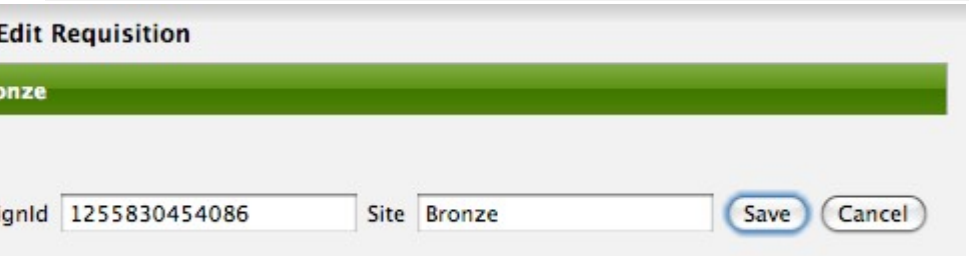
Clicking the *Add New Group* button will create the group and will redisplay the page including this new group among the list of any group(s) that have already been created.

At this point, the XML structure for holding the new provisioning group (a.k.a. an import requisition) has been persisted to the '$OPENNMS_ETC/imports/pending' directory.

Clicking the *Edit* link will bring you to the screen where you can begin the process of defining node entities that will be imported into OpenNMS Horizon. Click the Add Node button will begin the node entity creation process fill in the node label and click the *Save* button.

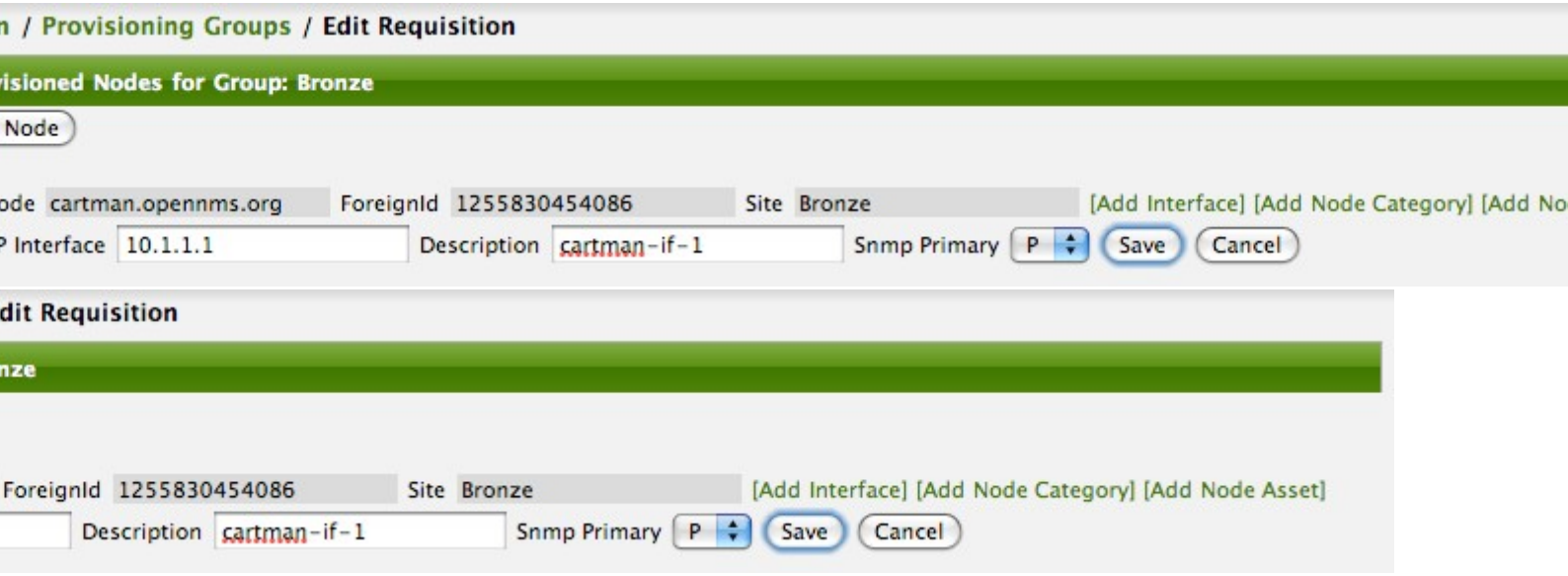
Creating a new Node definition in the Provisioning Group





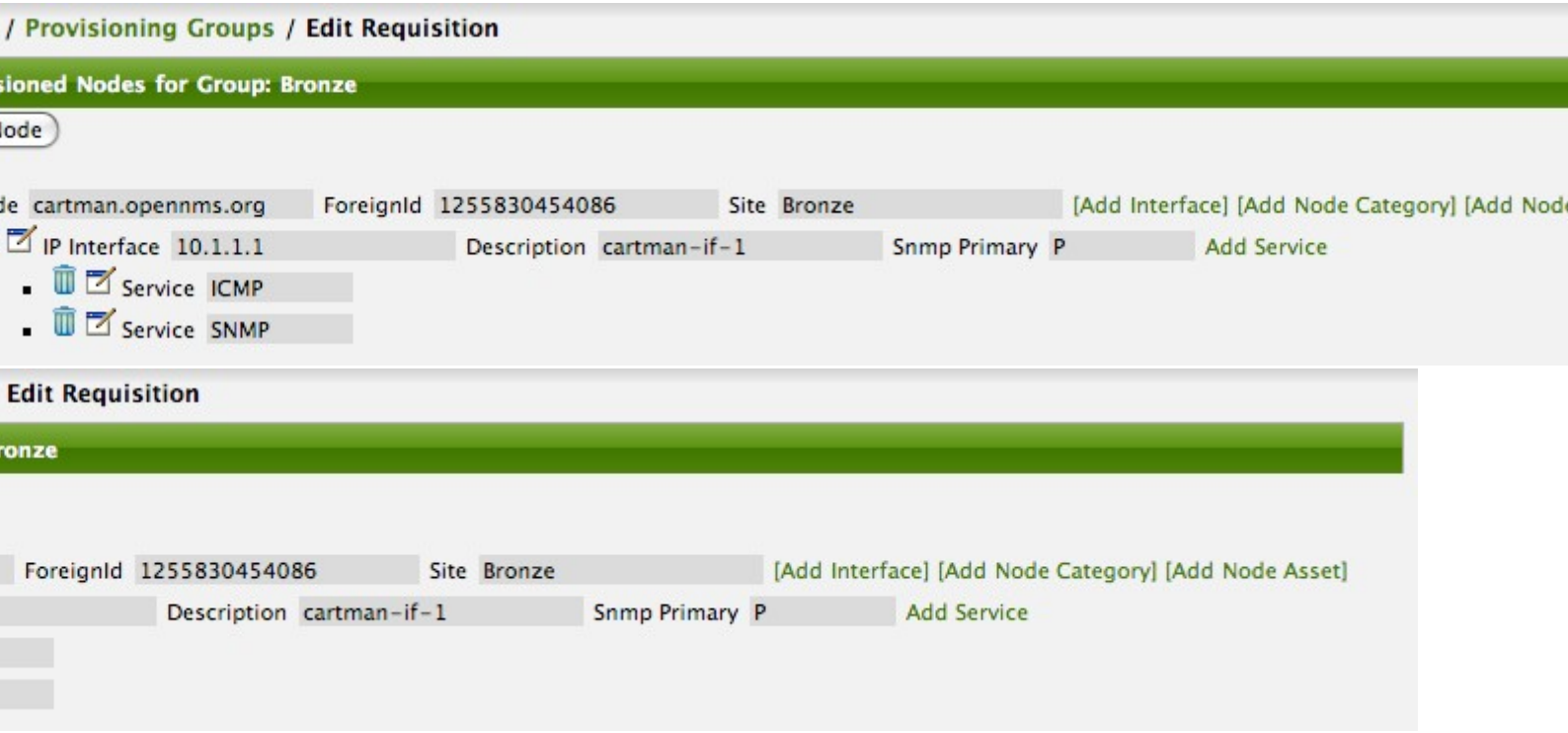
At this point, the provisioning group contains the basic structure of a node entity but it is not complete until the interface(s) and interface service(s) have been defined. After having clicked the *Save* button, as we did above presents, in the Web-UI, the options *Add Interface*, *Add Node Category*, and *Add Node Asset*. Click the *Add Interface* link to add an interface entity to the node.

Adding an Interface to the node definition



Enter the IP address for this interface entity, a description, and specify the Primary attribute as `P` (Primary), `S` (Secondary), `N` (Not collected), or `C` (Collected) and click the save button. Now the node entity has an interface for which services can be defined for which the Web-UI now presents the *Add Service* link. Add two services (ICMP, SNMP) via this link.

A complete node definition with all required elements defined.

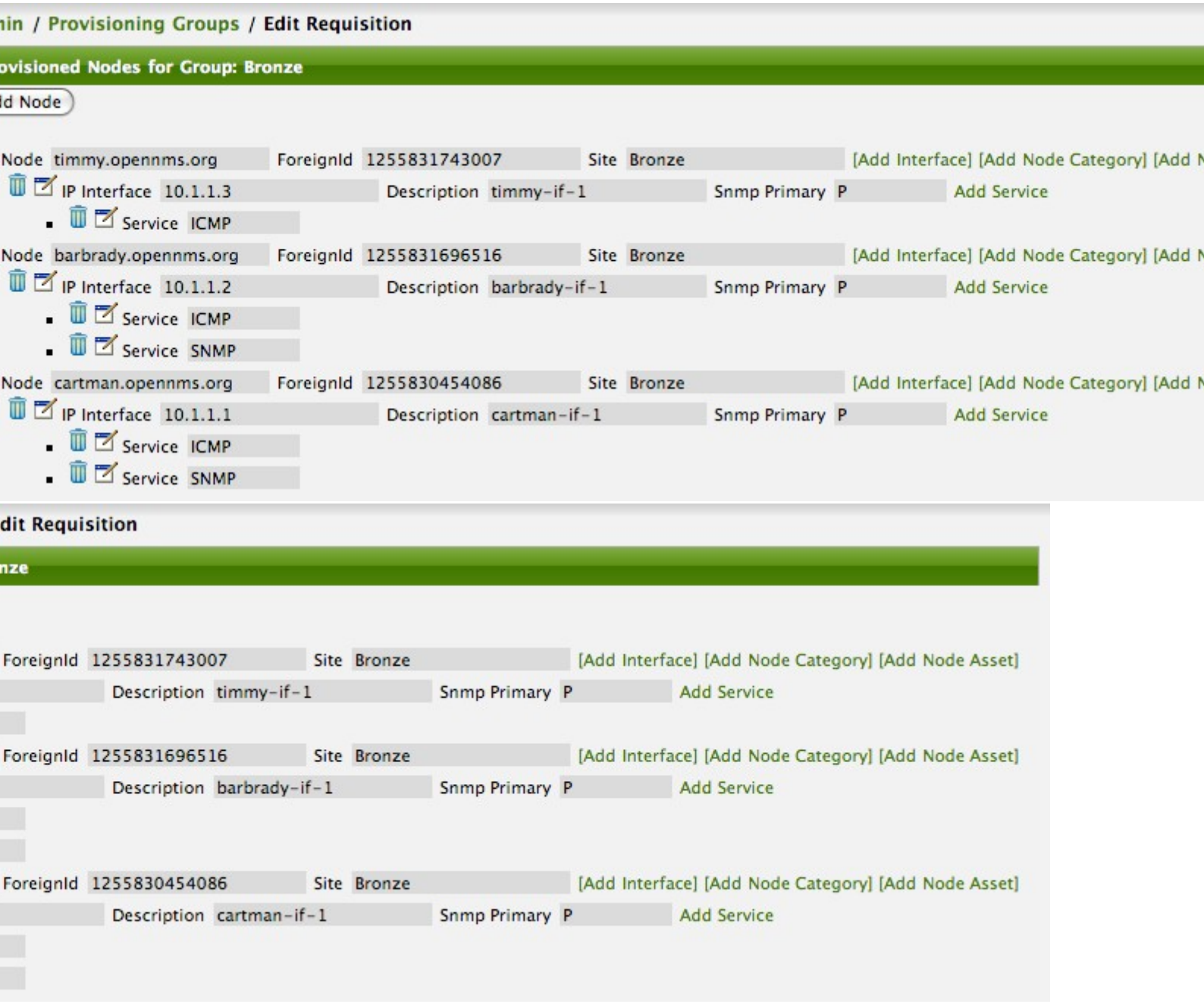


Now the node entity definition contains all the *required* elements necessary for importing this requisition into OpenNMS Horizon. At this point, all the interfaces that are required for the node

should be added. For example, NAT interfaces should be specified there are services that they provide because they will not be discovered during the Scan Phase.

Two more node definitions will be added for the benefit of this example.

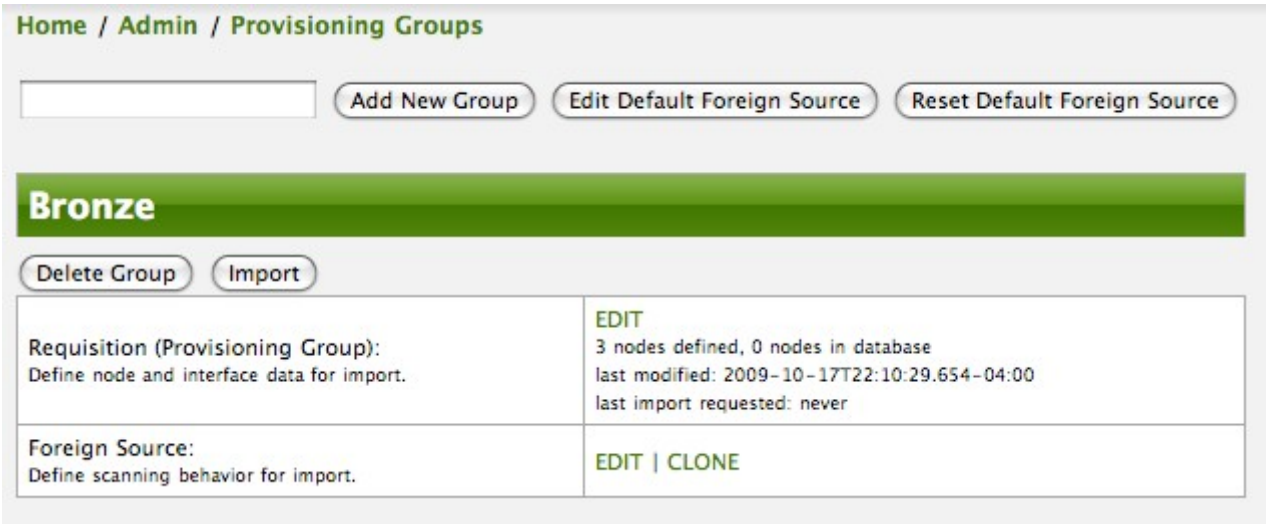
The completed requisition for the example Bronze Provisioning Group

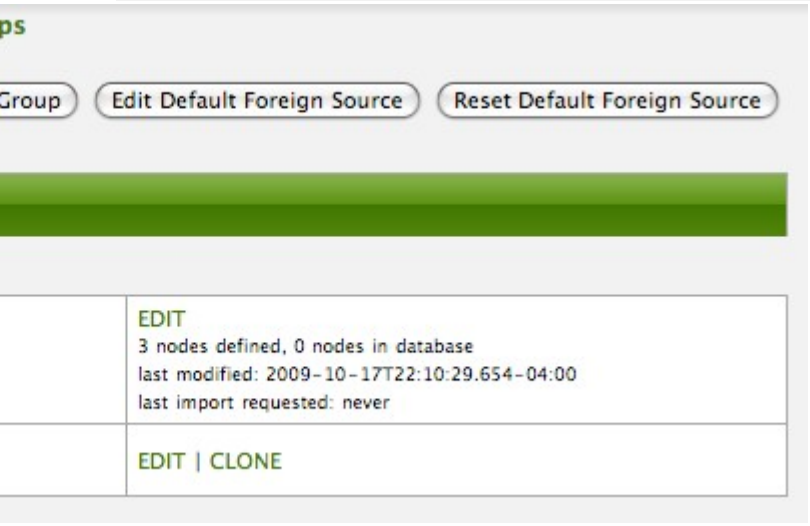


This set of nodes represents an import requisition for the *Bronze* provisioning group. As this requisition is being edited via the WebUI, changes are being persisted into the OpenNMS Horizon configuration directory '$OPENNMS_etc/imports/' pending as an XML file having the name `bronze.xml`.

The name of the XML file containing the import requisition is the same as the provisioning group name. Therefore naming your provisioning group without the use of spaces makes them easier to manage on the file system.

Click the *Done* button to return to the *Provisioning Groups* list screen. The details of the "Bronze" group now indicates that there are 3 nodes in the requisition and that there are no nodes in the DB from this group (a.k.a. foreign source). Additionally, you can see that time the requisition was last modified and the time it last imported are given (the time stamps are stored as attributes inside the requisition and are not the file system time stamps). These details are indicative of how well the DB represents what is in the requisition.





You can tell that this is a pending requisition for 2 reasons: 1) there are 3 nodes defined and 0 nodes in the DB, 2) the requisition has been modified since the last import (in this case *never*).

Import the Nodes

In this example, you see that there are 3 nodes in the pending requisition and 0 in the DB. Click the *Import* button to submit the requisition to the provisioning system (what actually happens is that the Web-UI sends an event to the Provisioner telling it to begin the Import Phase for this group).

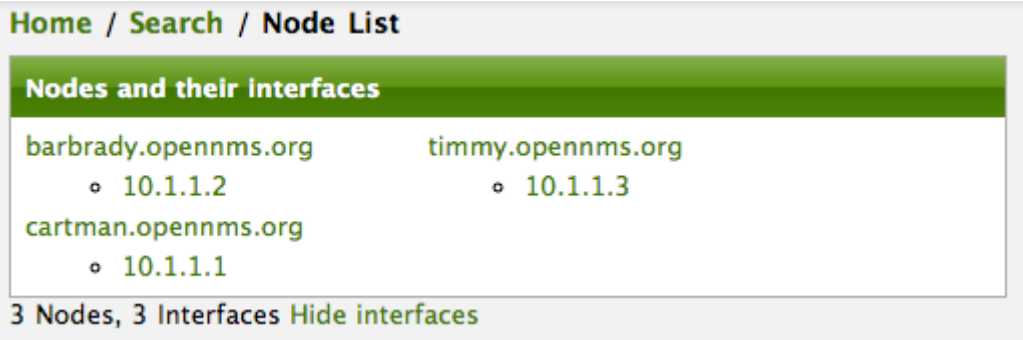
Do not refresh this page to check the values of these details. To refresh the details to verify the import, click the *Provisioning Groups* bread crumb item.

You should be able to immediately verify the importation of this provisioning group because the import happens very quickly. Provisiond has several threads ready for processing the import

operations of the nodes defined in this requisition.

A few SNMP packets are sent and received to get the SNMP details of the node and the interfaces defined in the requisition. Upon receipt of these packets (or not) each node is inserted as a DB transaction.

The nodes are now added to OpenNMS Horizon and are under management.



Following the import of a node with thousands of interfaces, you will be able to refresh the Interface table browser on the Node page and see that interfaces and services are being discovered and added in the background. This is the discovery component of directed discovery.

Adding a Node

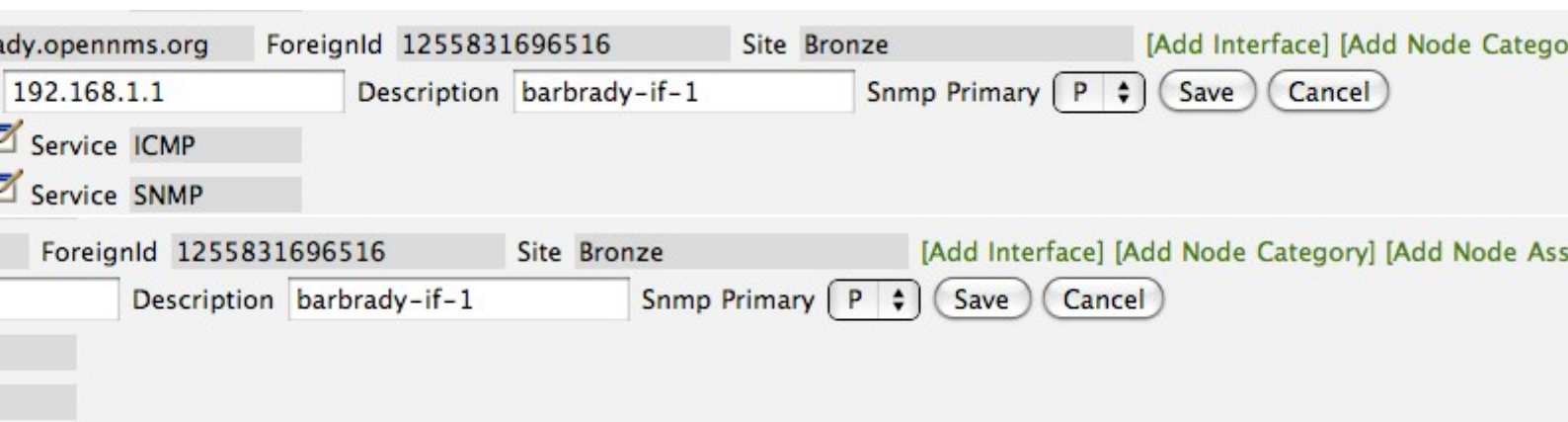
To direct that another node be added from a foreign source (in this example the Bronze Provisioning Group) simply add a new node definition and re-import. It is important to remember that all the node definitions will be re-imported and the existing managed nodes will be updated, if necessary.

Changing a Node

To direct changes to an existing node, simply add, change, or delete elements or attributes of the node definition and re- import. This is a great feature of having directed specific elements of a node in the requisition because that attributes will simply be changed. For example, to change the IP address of the Primary SNMP interface for the node, *barbrady.opennms.org*, just change the requisition and re-import.

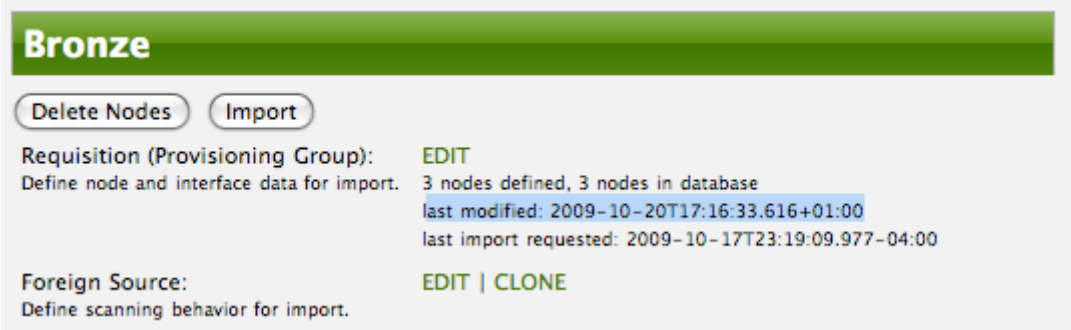
Each element in the Web-UI has an associated Edit icon Click this icon to change the IP address for barbrady.opennms.org, click save, and then Click the Done button.

Changing the IP address of barbrady.opennms.org from 10.1.1.2 to 192.168.1.1



The Web-UI will return you to the *Provisioning Groups* screen where you will see that there are the time stamp showing that the requisition's last modification is more recent that the last import time.

The Provisioning Group must be re-imported



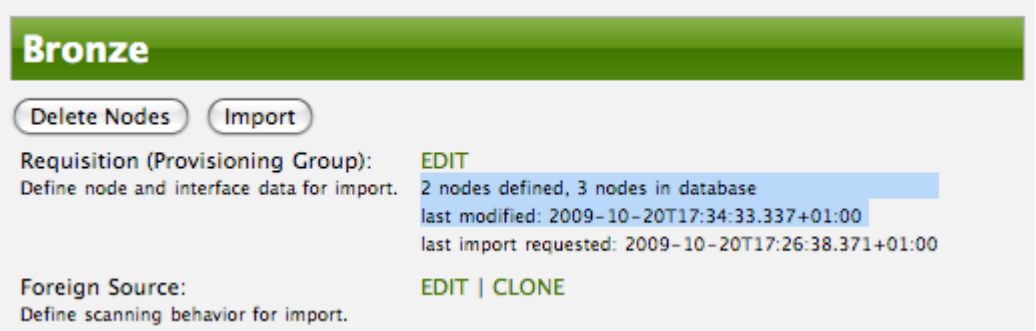
This provides an indication that the group must be re-imported for the changes made to the requisition to take effect. The IP Interface will be simply updated and all the required events (messages) will be sent to communicate this change within OpenNMS Horizon.

The IP interface for barbrady.opennms.org is immediately updated

Deleting a Node

Barbrady has not been behaving, as one might expect, so it is time to remove him from the system. Edit the provisioning group, click the delete button next to the node *barbrady.opennms.org*, click the *Done* button.

Bronze Provisioning Group definition indicates a node has been removed and requires an import to delete the node entity from the OpenNMS Horizon system



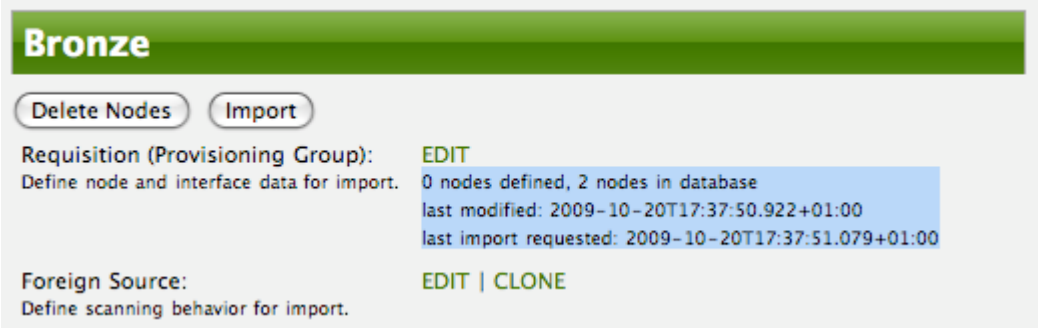
Click the Import button for the Bronze group and the Barbrady node and its interfaces, services, and any other related data will be immediately deleted from the OpenNMS Horizon system. All the required Events (messages) will be sent by Provisiond to provide indication to the OpenNMS Horizon system that the node Barbrady has been deleted.

Barbrady has been deleted

Deleting all the Nodes

There is a convenient way to delete all the nodes that have been provided from a specific foreign source. From the main *Admin/Provisioning Groups* screen in the Web-UI, click the *Delete Nodes* button. This button deletes all the nodes defined in the Bronze requisition. It is very important to note that once this is done, it cannot be undone! Well it can't be undone from the Web-UI and can only be undone if you've been good about keeping a backup copy of your '$OPENMS_ETC/' directory tree. If you've made a mistake, before you re-import the requisition, restore the `Bronze.xml` requisition from your backup copy to the '$OPENNMS_ETC/imports' directory.

All node definitions have been removed from the Bronze requisition. The Web-UI indicates an import is now required to remove them from OpenNMS Horizon.



Clicking the *Import* button will cause the *Audit Phase* of *Provisiond* to determine that all the nodes from the *Bronze* group (foreign source) should be deleted from the DB and will create *Delete* operations. At this point, if you are satisfied that the nodes have been deleted and that you will no longer require nodes to be defined in this Group, you will see that the *Delete Nodes* button has now changed to the *Delete Group* button. The *Delete Group* button is displayed when there are no nodes entities from that group (foreign source) in OpenNMS Horizon.

When no node entities from the group exist in OpenNMS Horizon, then the *Delete Group* button is displayed.

6.5.2. Advanced Provisioning Example

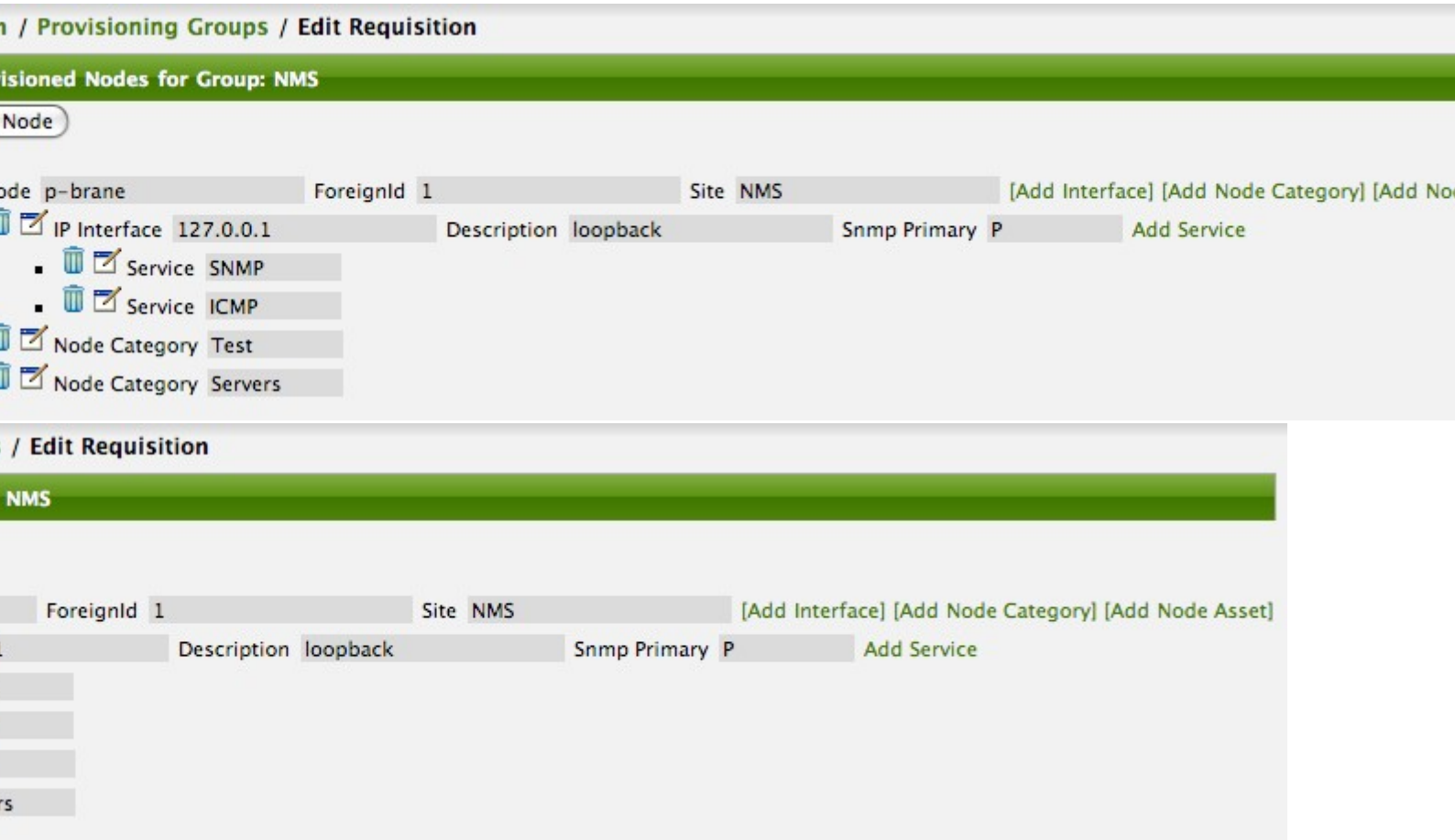
In the previous example, we provisioned 3 nodes and let *Provisiond* complete all of its import phases using a default foreign source definition. Each Provisioning Group can have a separate foreign source definition that controls:

- The rescan interval
- The services to be detected
- The policies to be applied

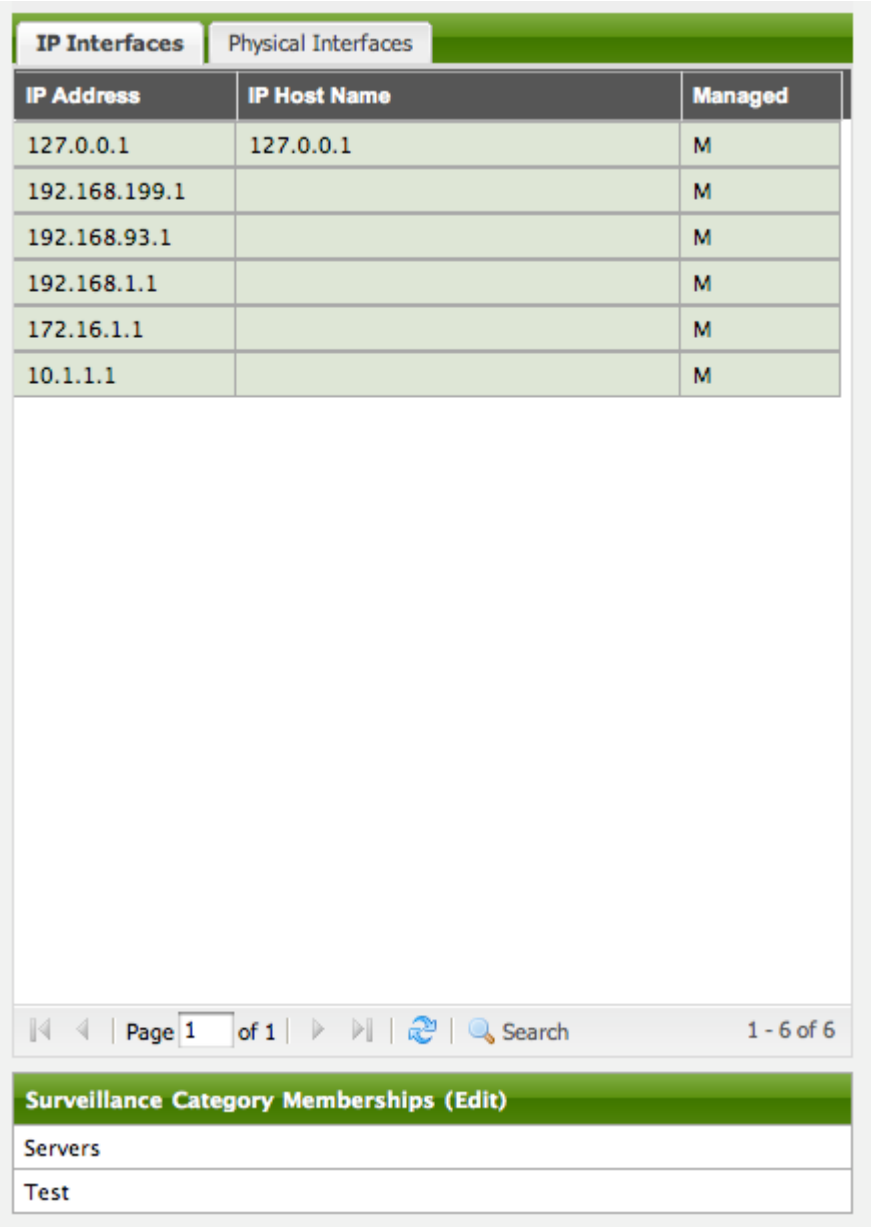
This example will demonstrate how to create a foreign source definition and how it is used to control the behavior of Provisiond when importing a *Provisioning Group/foreign source requisition*.

First let's simply provision the node and let the default foreign source definition apply.

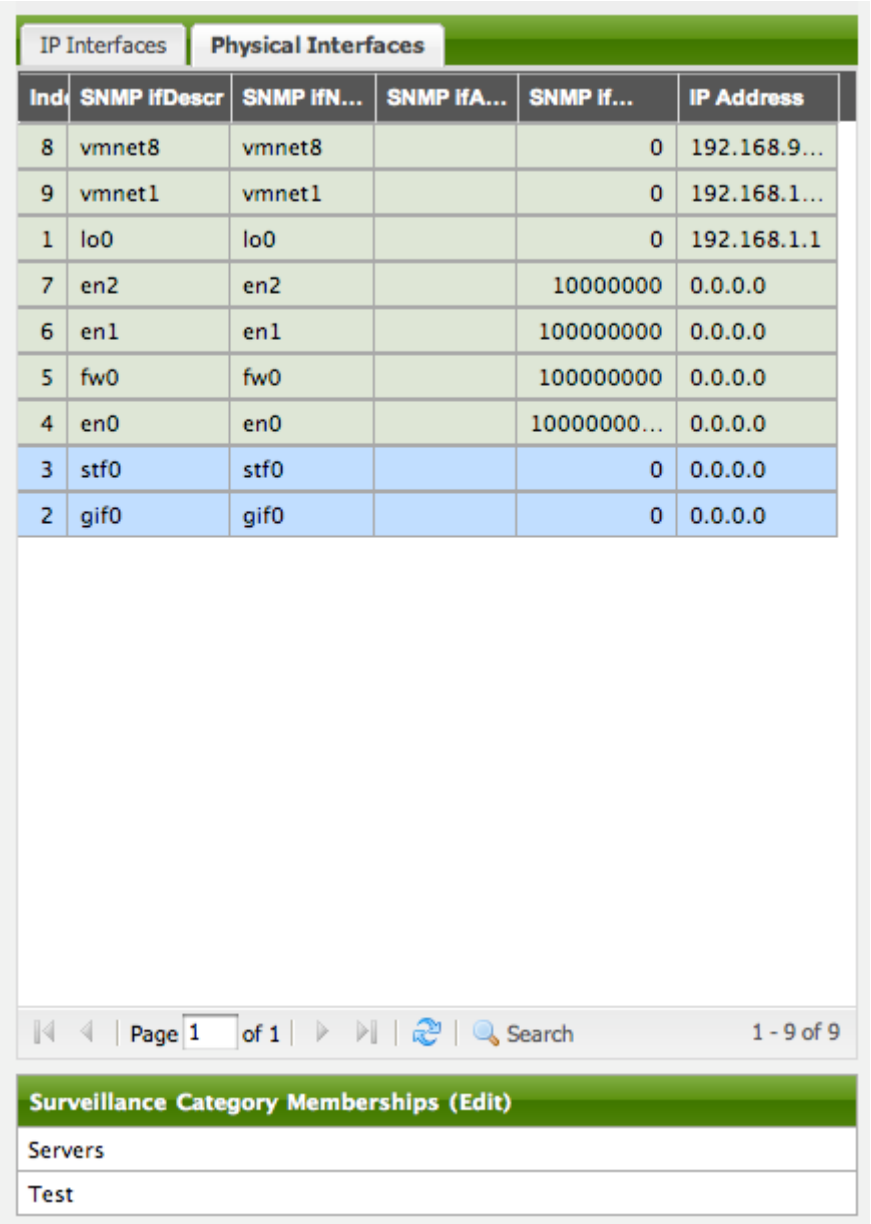
The node definition used for the Advanced Provisioning Example



Following the import, All the IP and SNMP interfaces, in addition to the interface specified in the requisition, have been discovered and added to the node entity. The default foreign source definition has no polices for controlling which interfaces that are discovered either get persisted or managed by OpenNMS Horizon.



Logical and Physical interface and Service entities directed and discovered by Provisiond.


IP Interfaces
Physical Interfaces
Ind
SNMP ifDescr
SNMP ifN...
SNMP ifA...
SNMP if...
IP Address
8 vmnet8 vmnet8 0 192.168.9...
9 vmnet1 vmnet1 0 192.168.1...
1 lo0 lo0 0 192.168.1.1
7 en2 en2 10000000 0.0.0.0
6 en1 en1 100000000 0.0.0.0
5 fw0 fw0 100000000 0.0.0.0
4 en0 en0 10000000... 0.0.0.0
3 stf0 stf0 0 0.0.0.0
2 gif0 gif0 0 0.0.0.0
Page 1 of 1
Search
1 - 9 of 9
Surveillance Category Memberships (Edit)
Servers
Test

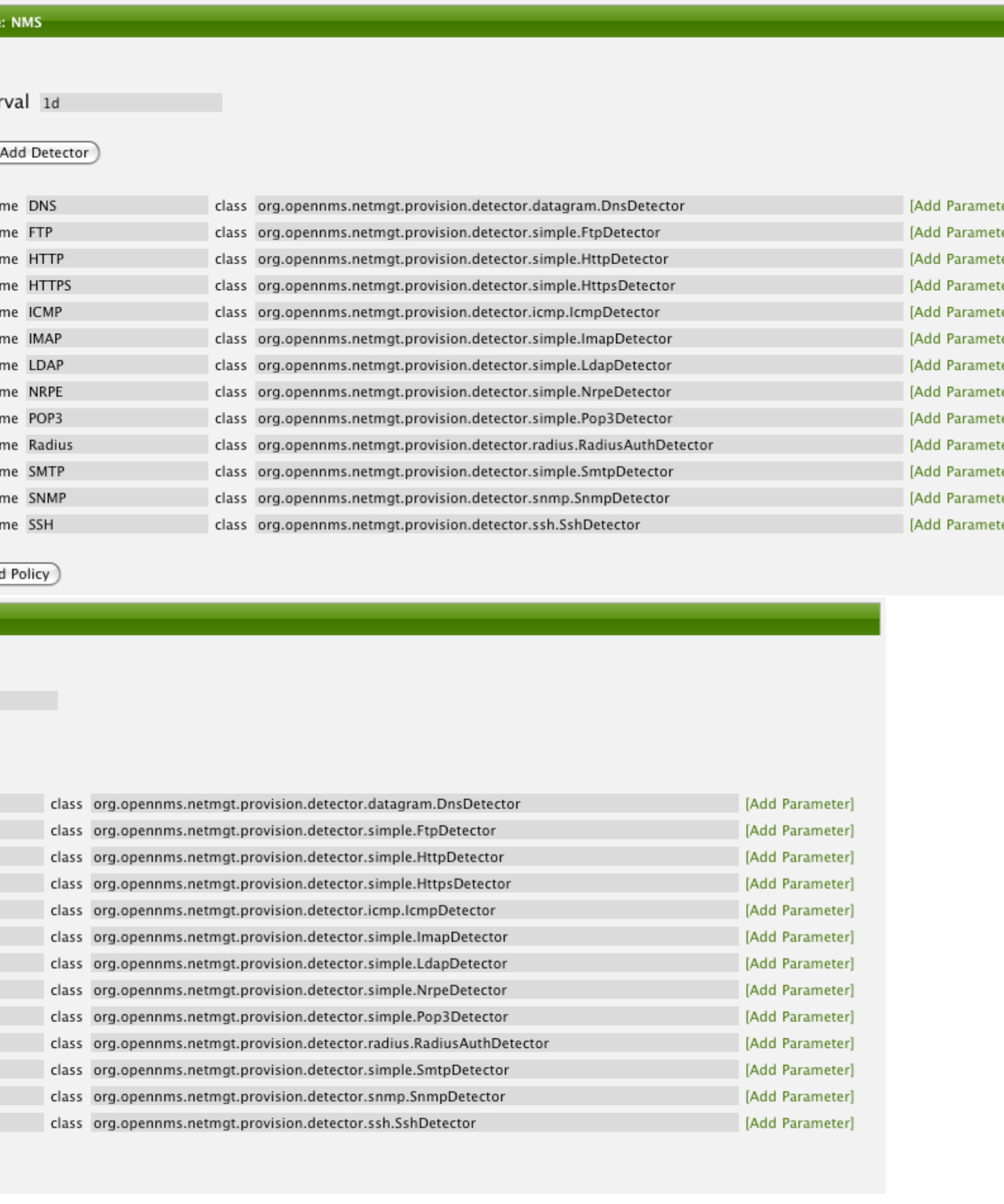
Service Detection

As IP interfaces are found during the node scan process, service detection tasks are scheduled for each IP interface. The service detections defined in the foreign source determines which services are to be detected and how (i.e. the values of the parameters that parameters control how the service is detected, port, timeout, etc.).

Applying a New Foreign Source Definition

This example node has been provisioned using the Default foreign source definition. By navigating to the Provisioning Groups screen in the OpenNMS Horizon Web-UI and clicking the Edit Foreign Source link of a group, you can create a new foreign source definition that defines service detection and policies. The policies determine entity persistence and/or set attributes on the discovered entities that control OpenNMS Horizon management behaviors.

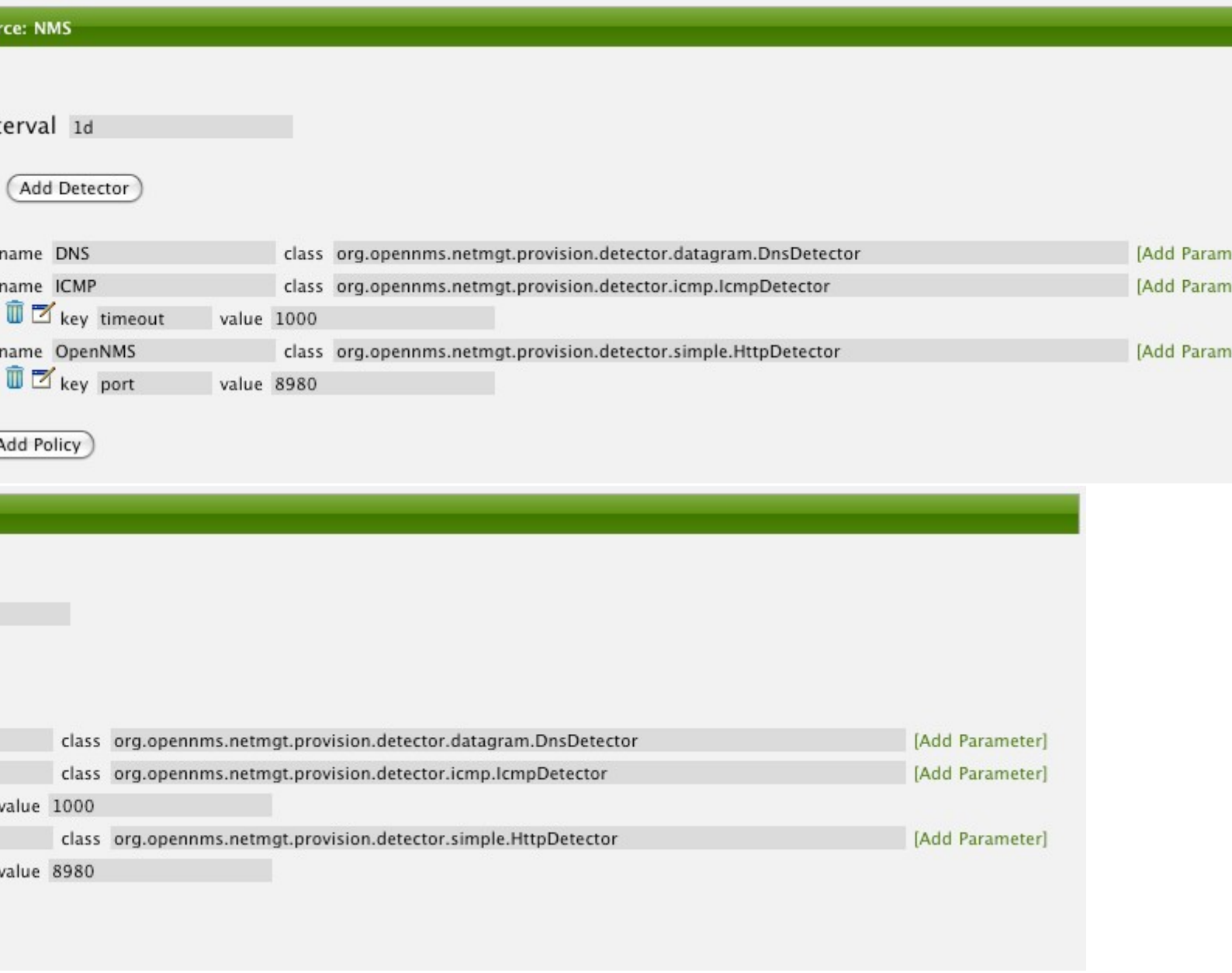
When creating a new foreign source definition, the default definition is used as a template.



In this UI, new Detectors can be added, changed, and removed. For this example, we will remove detection of all services accept ICMP and DNS, change the timeout of ICMP detection, and a new

Service detection for OpenNMS Horizon Web-UI.

Custom foreign source definition created for NMS Provisioning Group (foreign source).



Click the Done button and re-import the NMS Provisioning Group. During this and any subsequent re-imports or re- scans, the OpenNMS Horizon detector will be active, and the detectors that have been removed will no longer test for the related services for the interfaces on nodes managed in the provisioning group (requisition), however, the currently detected services will not be removed. There are 2 ways to delete the previously detected services:

1. Delete the node in the provisioning group, re-import, define it again, and finally re-import again
2. Use the ReST API to delete unwanted services. Use this command to remove each unwanted service from each interface, iteratively:

```
curl -X DELETE -H "Content-Type: application/xml" -u admin:admin
http://localhost:8980/opennms/rest/nodes/6/ipinterfaces/172.16.1.1/services/DNS
```

There is a sneaky way to do #1. Edit the provisioning group and just change the foreign ID. That will make Provisiond think that a node was deleted and a new node was added in the same requisition! Use this hint with caution and an full understanding of the impact of deleting an existing node.

Provisioning with Policies

The Policy API in Provisiond allow you to control the persistence of discovered IP and SNMP Interface entities and Node Categories during the Scan phase.

Matching IP Interface Policy

The Matching IP Interface policy controls whether discovered I interfaces are to be persisted and if they are to be persisted, whether or not they will be forced to be Managed or Unmanaged.

Continuing with this example Provisioning Group, we are going to define a few policies that:

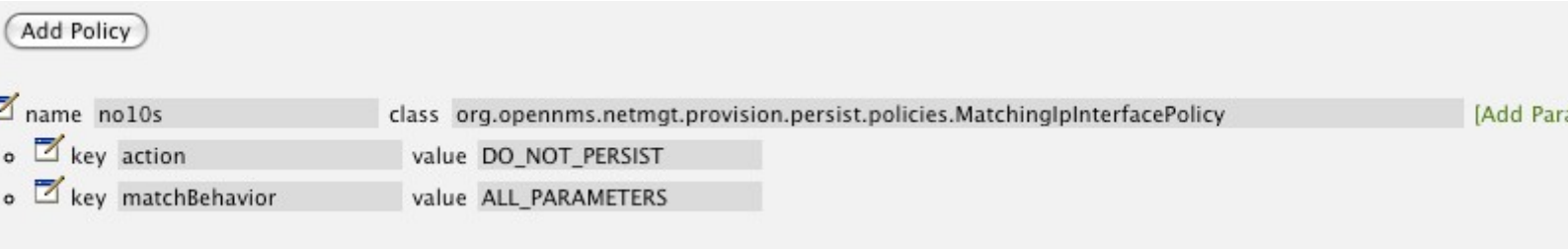
a. Prevent discovered 10 network addresses from being persisted

b. Force 192.168 network addresses to be unmanaged

From the foreign source definition screen, click the Add Policy button and you the definition of a new policy will begin with a field for naming the policy and a drop down list of the currently installed policies. Name the policy *no10s*, make sure that the *Match IP Interface policy* is specified in the class list and click the Save button. This action will automatically add all the parameters required for the policy.

The two required parameters for this policy are action and matchBehavior.

The action parameter can be set to DO_NOT_PERSIST, Manage, or UnManage.



Creating a policy to prevent persistence of 10 network IP interfaces.

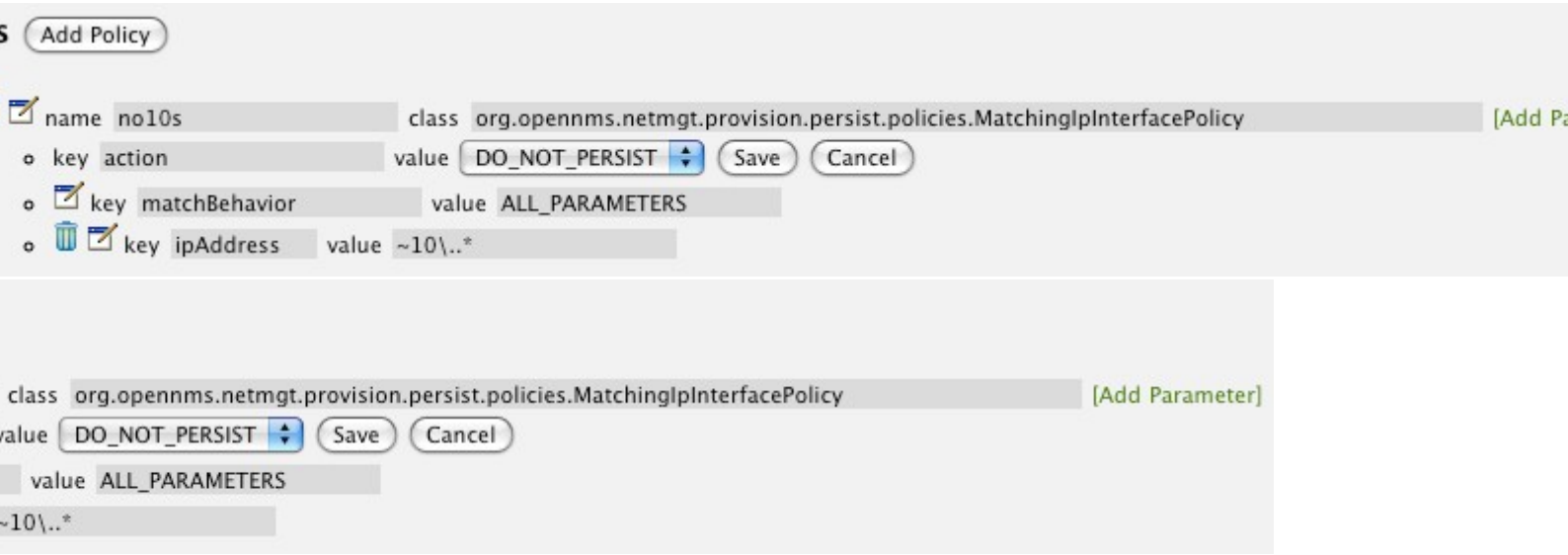
The *DO_NOT_PERSIST* action does just what it indicates, it prevents discovered IP interface entities from being added to OpenNMS Horizon when the *matchBehavior* is satisfied. The Manage and UnManage values for this action allow the IP interface entity to be persisted by control whether or not that interface should be managed by OpenNMS Horizon.

The matchBehavior action is a boolean control that determines how the optional parameters will

be evaluated. Setting this parameter's value to *ALL_PARAMETERS* causes *Provisiond* to evaluate each optional parameter with boolean *AND* logic and the value *ANY_PARAMETERS* will cause *OR* logic to be applied.

Now we will add one of the optional parameters to filter the 10 network addresses. The Matching IP Interface policy supports two additional parameters, *hostName* and *ipAddress*. Click the *Add Parameter* link and choose *ipAddress* as the *key*. The *value* for either of the optional parameters can be an exact or regular expression match. As in most configurations in OpenNMS Horizon where regular expression matching can be optionally applied, prefix the value with the `~` character.

Example Matching IP Interface Policy to not Persist 10 Network addresses

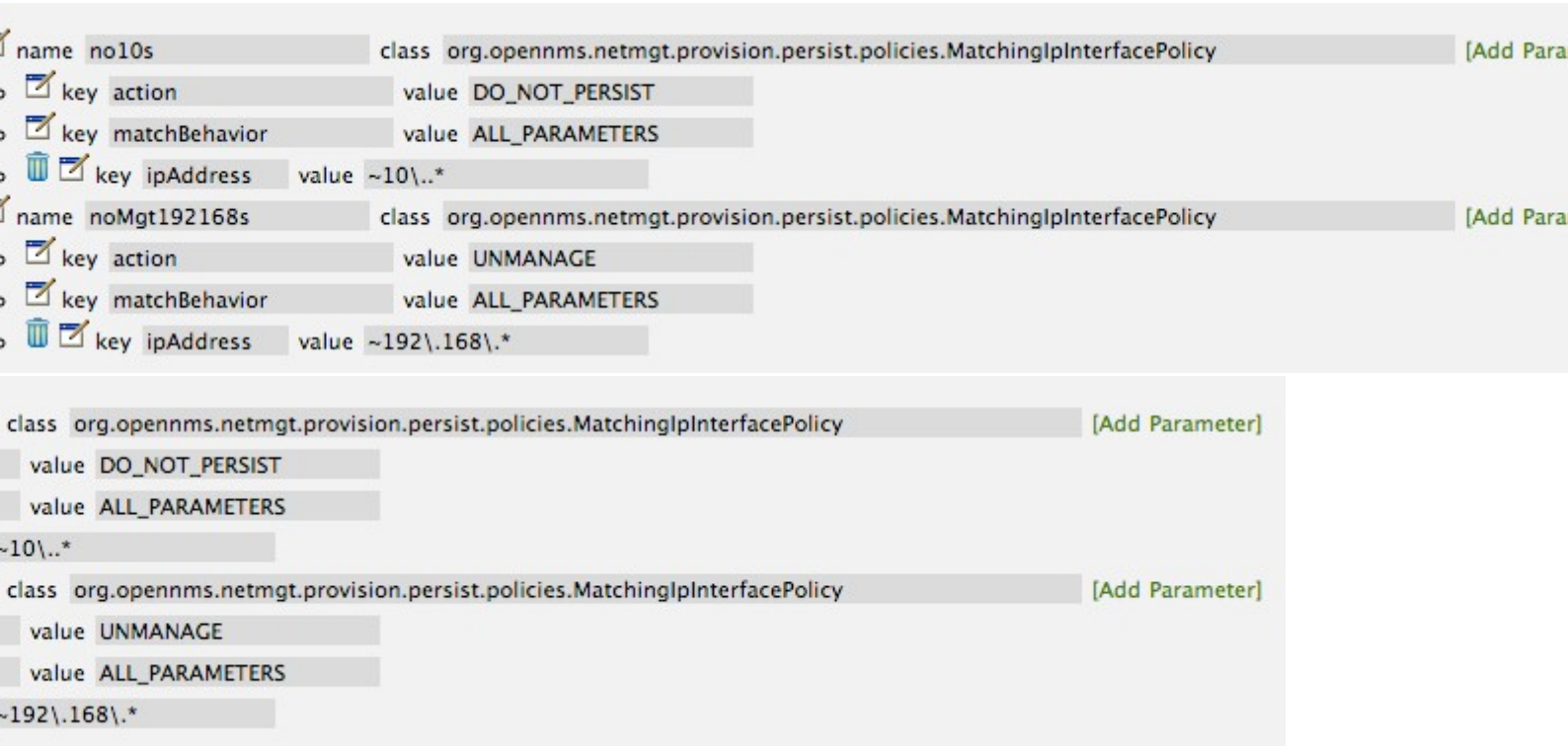


Any subsequent scan of the node or re-imports of NMS provisioning group will force this policy to be applied. IP Interface entities that already exist that match this policy will not be deleted. Existing interfaces can be deleted by recreating the node in the *Provisioning Groups* screen (simply change the foreign ID and re-import the group) or by using the ReST API:

```
curl -X DELETE -H "Content-Type: application/xml" -u admin:admin
http://localhost:8980/opennms/rest/nodes/6/ipinterfaces/10.1.1.1
```

The next step in this example is to define a policy that sets discovered 192.168 network addresses to be unmanaged (not managed) in OpenNMS Horizon. Again, click the Add Policy button and let's call this policy *noMgt192168s*. Again, choose the Mach IP Interface policy and this time set the action to *UNMANAGE*.

Policy to not manage IP interfaces from 192.168 networks



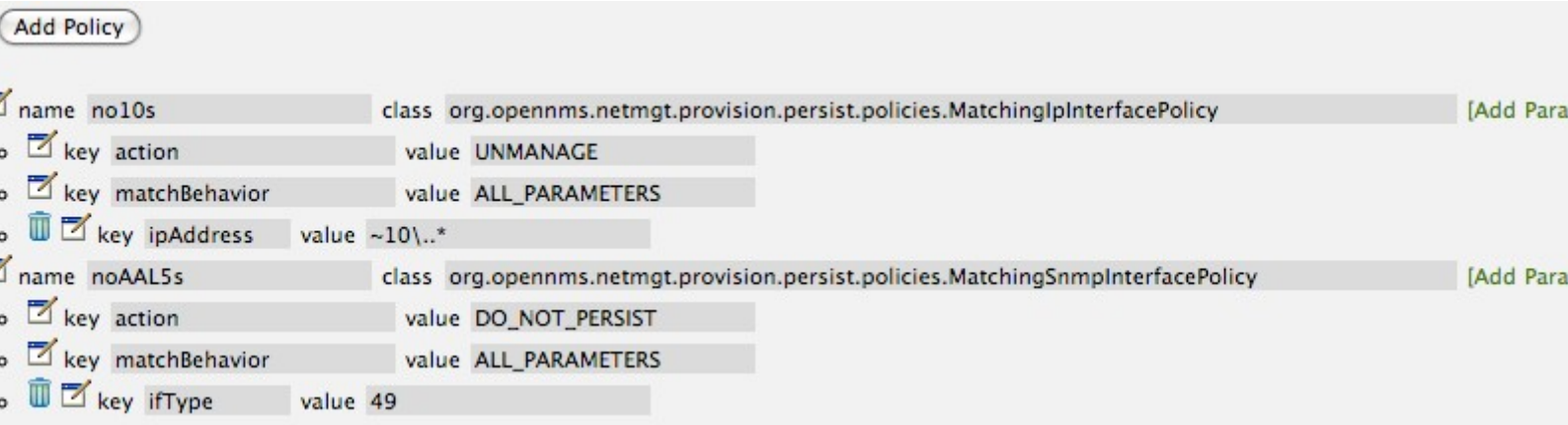
The *UNMANAGE* behavior will be applied to existing interfaces.

Matching SNMP Interface Policy

Like the Matching IP Interface Policy, this policy controls the whether discovered SNMP interface entities are to be persisted and whether or not OpenNMS Horizon should collect performance metrics from the SNMP agent for Interface's index (MIB2 IfIndex).

In this example, we are going to create a policy that doesn't persist interfaces that are *AAL5* over *ATM* or type *49* (*ifType*). Following the same steps as when creating an IP Management Policy, edit the foreign source definition and create a new policy. Let's call it: *noAAL5s*. We'll use Match SNMP Interface class for each policy and add a parameter with *ifType* as the key and *49* as the value.

Matching SNMP Interface Policy example for Persistence and Data Collection

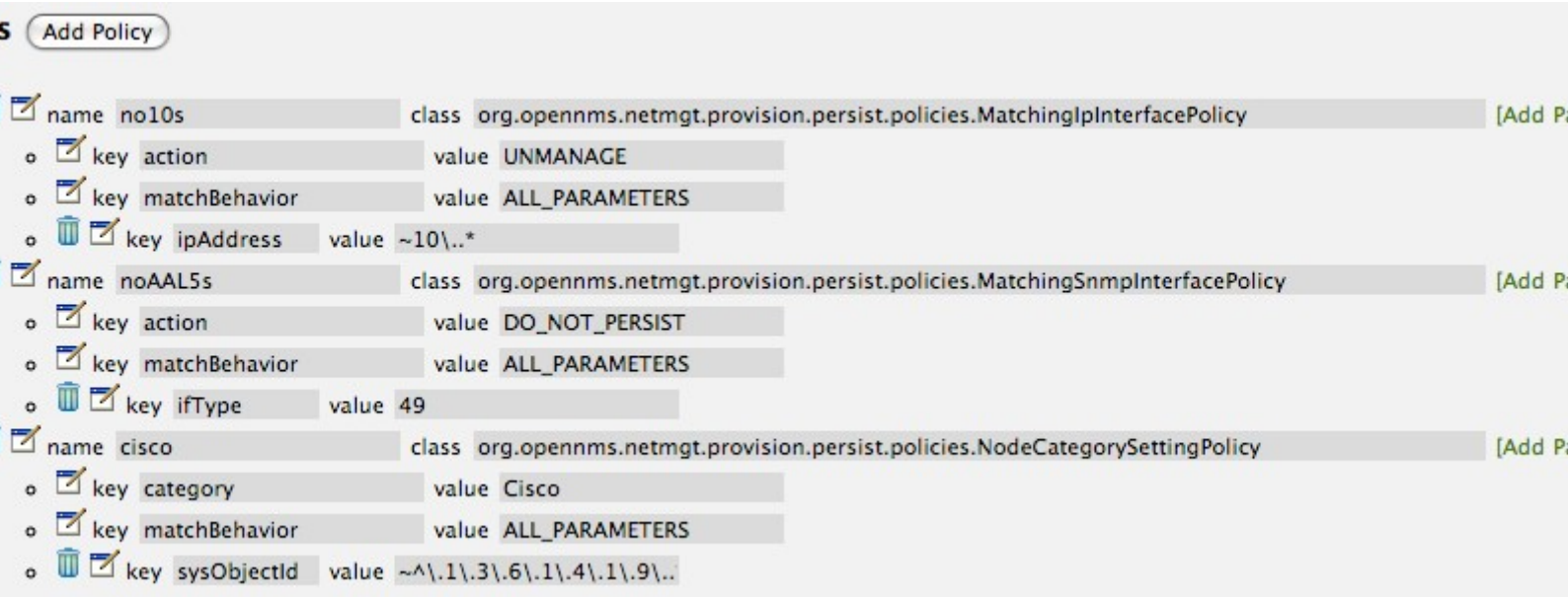


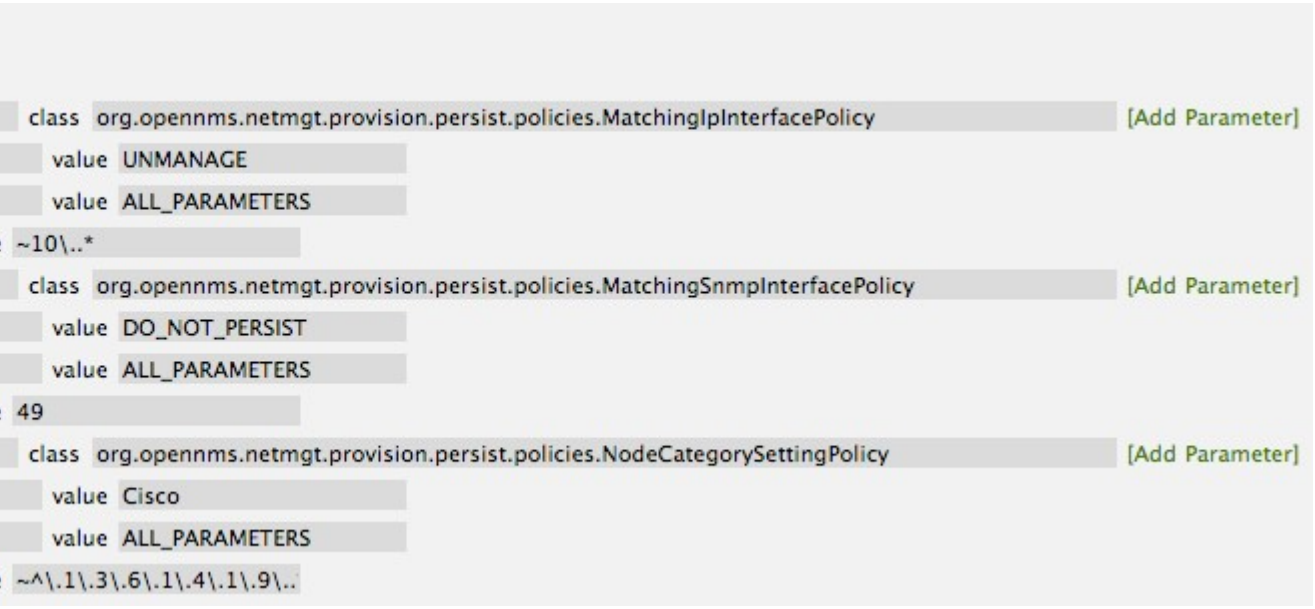
At the appropriate time during the scanning phase, Provisiond will evaluate the policies in the foreign source definition and take appropriate action. If during the policy evaluation process any policy matches for a “DO_NOT_PERSIST” action, no further policy evaluations will happen for that particular entity (IP Interface, SNMP Interface).

Node Categorization Policy

With this policy, nodes entities will automatically be assigned categories. The policy is defined in the same manner as the IP and SNMP interface polices. Click the Add Policy button and give the policy name, `cisco` and choose the *Set Node Category* class. Edit the required *category* key and set the value to `Cisco`. Add a policy parameter and choose the *sysObjectId* key with a value `~^\.1\.3\.6\.1\.4\.1\.9\..*`.

Example: Node Category setting policy





New Import Capabilities

Several new XML entities have been added to the import requisition since the introduction of the OpenNMS Importer service in version 1.6. So, in addition to provisioning the basic node, interface, service, and node categories, you can now also provision asset data.

Provisiond Configuration

The configuration of the Provisioning system has moved from a properties file (`model-importer.properties`) to an XML based configuration container. The configuration is now extensible to allow the definition of 0 or more import requisitions each with their own *Cron* based schedule for automatic importing from various sources (intended for integration with external URL such as HTTP and this new DNS protocol handler.

A default configuration is provided in the OpenNMS Horizon `etc/` directory and is called: `provisiond-configuration.xml`. This default configuration has an example for scheduling an import from a DNS server running on the localhost requesting nodes from the zone, localhost and will be imported once per day at the stroke of midnight. Not very practical but is a good example.

```
<?xml version="1.0" encoding="UTF-8"?>
    <provisiond-configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.opennms.org/xsd/config/provisiond-configuration"
        foreign-source-dir="/opt/opennms/etc/foreign-sources"
        requistion-dir="/opt/opennms/etc/imports"
        importThreads="8"
        scanThreads="10"
        rescanThreads="10"
        writeThreads="8" >
    <!--
        http://www.quartz-scheduler.org/documentation/quartz-
1.x/tutorials/crontrigger[http://www.quartz-scheduler.org/documentation/quartz-
1.x/tutorials/crontrigger]
        Field Name Allowed Values Allowed Special Characters
        Seconds 0-59 , - * / Minutes 0-59 , - * / Hours 0-23 , - * /
        Day-of-month1-31, - * ? / L W C Month1-12 or JAN-DEC, - * /
        Day-of-Week1-7 or SUN-SAT, - * ? / L C # Year (Opt)empty, 1970-2099, - * /
    -->

    <requisition-def import-name="NMS"
                     import-url-resource="file://opt/opennms/etc/imports/NMS.xml">
        <cron-schedule>0 0 0 * * ? *</cron-schedule> <!-- daily, at midnight -->
    </requisition-def>
</provisiond-configuration>
```

Configuration Reload

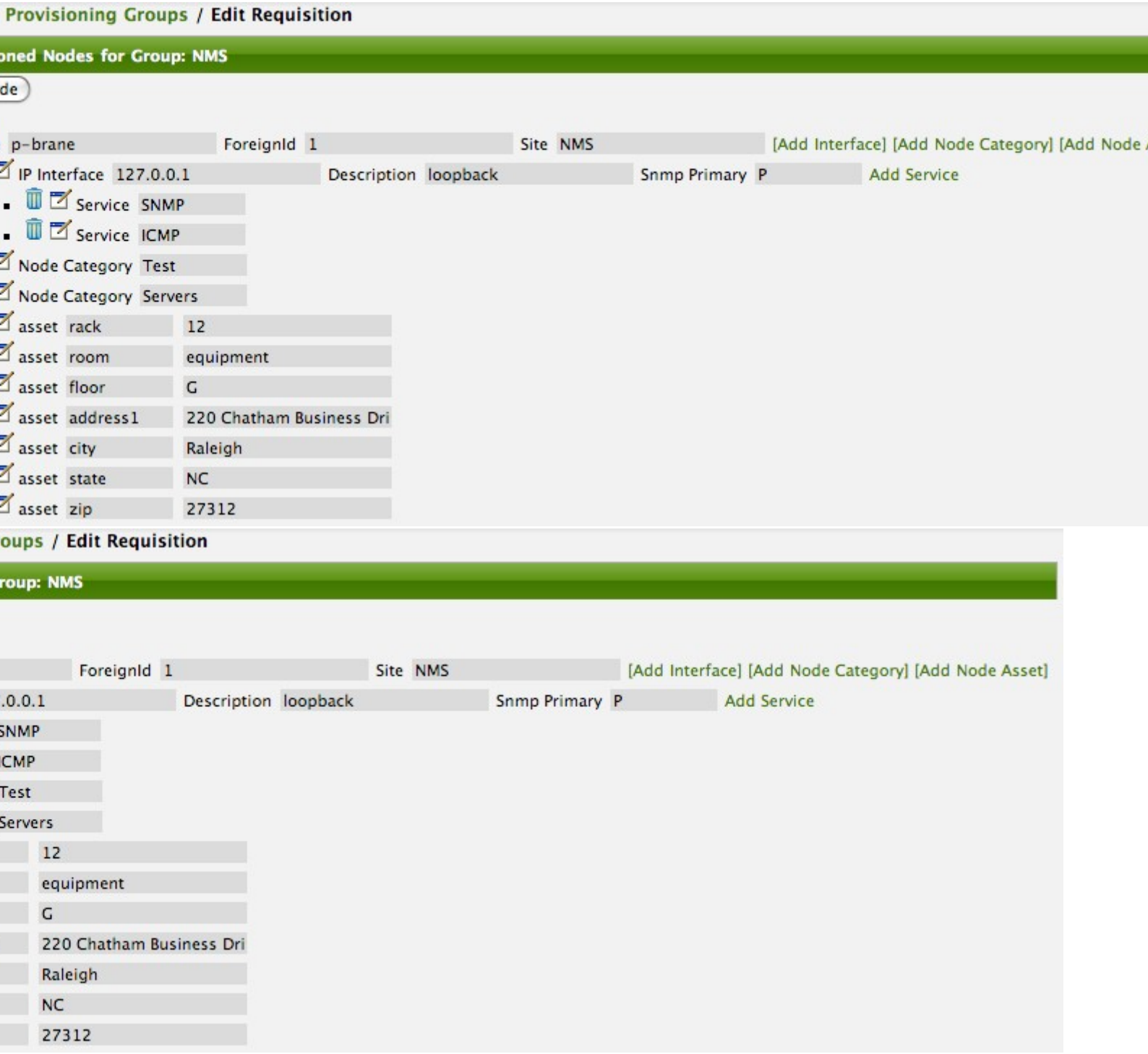
Like many of the daemon configurations in the 1.7 branch, *Provisiond's* configuration is re-loadable without having to restart OpenNMS. Use the reloadDaemonConfig uei:

```
/opt/opennms/bin/send-event.pl uei.opennms.org/internal/reloadDaemonConfig --parm
'daemonName Provisiond'
```

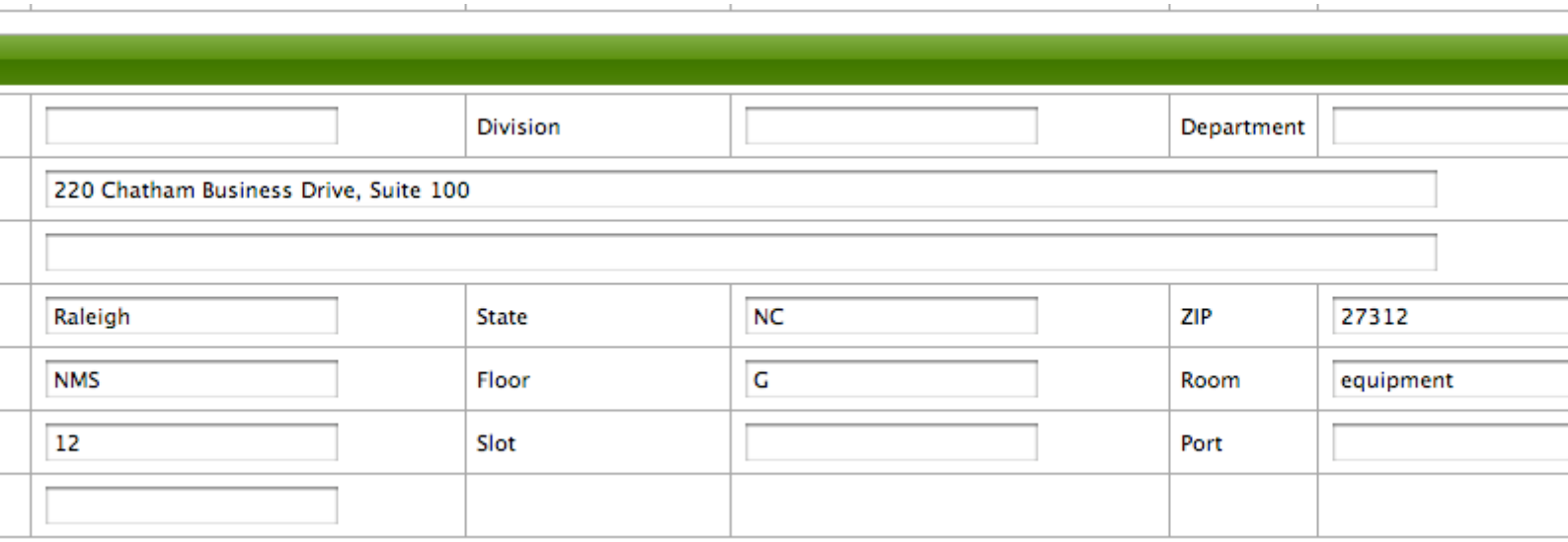
This means that you don't have to restart OpenNMS Horizon every time you update the configuration!

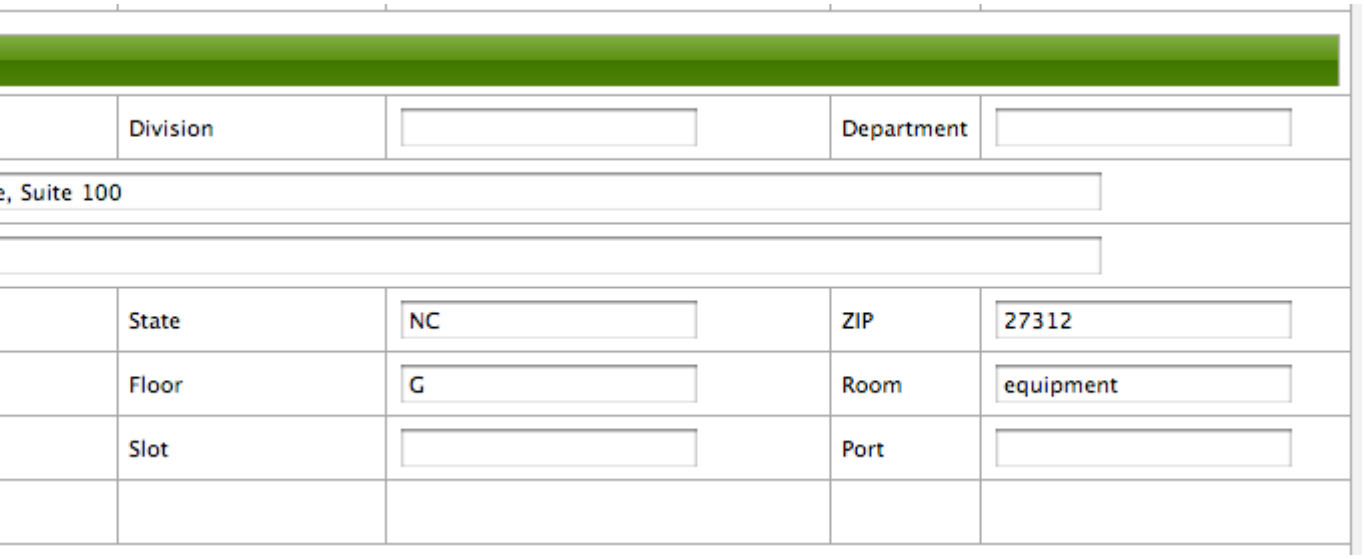
Provisioning Asset Data

The Provisioning Groups Web-UI had been updated to expose the ability to add Node Asset data in an import requisition. Click the *Add Node Asset* link and you can select from a drop down list all the possible node asset attributes that can be defined.



After an import, you can navigate to the *Node Page* and click the *Asset Info* link and see the asset data that was just provided in the requisition.





External Requisition Sources

Because Provisiond takes a *URL* as the location service for import requisitions, OpenNMS Horizon can be easily extended to support sources in addition to the native URL handling provided by Java: *file://*, *http://*, and *https://*. When you configure *Provisiond* to import requisitions on a schedule you specify using a *URL* Resource. For requisitions created by the *Provisioning Groups* WebUI, you can specify a file based URL.

<need further documentation>

Provisioning Nodes from DNS

The new Provisioning service in OpenNMS Horizon is continuously improving and adapting to the needs of the community. One of the most recent enhancements to the system is built upon the very flexible and extensible API of referencing an import requisition's location via a URL. Most commmonly, these URLs are files on the file system (i.e. `file:/opt/opennms/etc/imports/<my-provisioning-group.xml>`) as requisitions created by the Provisioning Groups UI. However, these same requistions for adding, updating, and deleting nodes (based on the original model importer) can also come from URLs specifying the HTTP protocol: http://myinventory.server.org/nodes.cgi)

Now, using Java's extensible protocol handling specification, a new protocol handler was created so

that a URL can be specified for requesting a Zone Transfer (*AXFR*) request from a DNS server. The *A records* are recorded and used to build an import requisition. This is handy for organizations that use DNS (possibly coupled with an IP management tool) as the data base of record for nodes in the network. So, rather than ping sweeping the network or entering the nodes manually into OpenNMS Horizon Provisioning UI, nodes can be managed via 1 or more DNS servers. The format of the URL for this new protocol handler is:

```
dns://<host>[:port]/<zone>[/<foreign-source>/][?expression=<regex>]
```

Simple Example

```
dns://my-dns-server/myzone.com
```

This will import all *A records* from the host *my-dns-server* on port 53 (default port) from zone *myzone.com* and since the foreign source (a.k.a. the provisioning group) is not specified it will default to the specified zone.

Using a Regular Expression Filter

You can also specify a subset of the *A records* from the zone transfer using a regular expression:

```
dns://my-dns-server/myzone.com/portland/?expression=^por-.*
```

This will import all nodes from the same server and zone but will only manage the nodes in the zone matching the regular expression `^port-.*` and will and they will be assigned a unique foreign source (provisioning group) for managing these nodes as a subset of nodes from within the specified zone.

URL Encoding

If your expression requires URL encoding (for example you need to use a `?` in the expression) it must be properly encoded.

```
dns://my-dns-server/myzone.com/portland/?expression=^por[0-9]%3F
```

DNS Setup

Currently, the DNS server requires to be setup to allow a zone transfer from the OpenNMS Horizon server. It is recommended that a secondary DNS server is running on OpenNMS Horizon and that the OpenNMS Horizon server be allowed to request a zone transfer. A quick way to test if zone transfers are working is:

```
dig -t AXFR @<dn5Server> <zone>
```

6.6. Adapters

The OpenNMS Horizon *Provisiond API* also supports *Provisioning Adapters* (plugins) for integration with external systems during the provisioning Import phase. When node entities are added, updated, deleted, or receive a configuration management change event, OpenNMS Horizon will call the adapter for the provisioning activities with integrated systems.

Currently, OpenNMS Horizon supports the following adapters:

6.6.1. DDNS Adapter

The Opposite end of *Provisiond* integration from the DNS Requisition Import, is the *DDNS adapter*. This adapter uses the *dynamic DNS protocol* to update a DNS system as nodes are provisioned into OpenNMS Horizon. To configure this adapter, edit the `opennms.properties` file and set the `importer.adapter.dns.server property`:

```
importer.adapter.dns.server=192.168.1.1
```

6.6.2. RANCID Adapter

Integration has been integrated with RANCID though this new API.

<More documentation needed>

Maps (soon to be moved to Mapd) <documentation required>

WiMax-Link (soon to be moved to Linkd) <documentation required>

6.7. Integrating with Provisiond

The ReST API should be used for integration from other provisioning systems with OpenNMS Horizon. The ReST API provides an interface for defining foreign sources and requisitions.

6.7.1. Provisioning Groups of Nodes

Just as with the WebUI, groups of nodes can be managed via the ReST API from an external system. The steps are:

1. Create a Foreign Source (if not using the default) for the group
2. Update the SNMP configuration for each node in the group
3. Create/Update the group of nodes

6.7.2. Example

Step 1 - Create a Foreign Source

If policies for this group of nodes are going to be specified differently than the default policy, then a foreign source should be created for the group. Using the ReST API, a foreign source can be provided. Here is an example:

The XML can be imbedded in the `curl` command option `-d` or be referenced from a file if the `@` prefix is used with the file name as in this case.

The XML file: `customer-a.foreign-source.xml`:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<foreign-source date-stamp="2009-10-12T17:26:11.616-04:00" name="customer-a" xmlns=
"http://xmlns.opennms.org/xsd/config/foreign-source">
    <scan-interval>1d</scan-interval>
    <detectors>
        <detector class="org.opennms.netmgt.provision.detector.icmp.IcmpDetector"
name="ICMP"/>
        <detector class="org.opennms.netmgt.provision.detector.snmp.SnmpDetector"
name="SNMP"/>
    </detectors>
    <policies>
        <policy class=
"org.opennms.netmgt.provision.persist.policies.MatchingIpInterfacePolicy" name="no-
192-168">
            <parameter value="UNMANAGE" key="action"/>
            <parameter value="ALL_PARAMETERS" key="matchBehavior"/>
            <parameter value="~^192\.168\..*" key="ipAddress"/>
        </policy>
    </policies>
</foreign-source>
```

Here is an example `curl` command used to create the foreign source with the above foreign source specification above:

```
curl -v -u admin:admin -X POST -H 'Content-type: application/xml' -d '@customer-
a.foreign-source.xml' http://localhost:8980/opennms/rest/foreignSources
```

Now that you've created the foreign source, it needs to be deployed by Provisiond. Here an the example using the `curl` command to deploy the foreign source:

```
curl -v -u admin:admin
http://localhost:8980/opennms/rest/foreignSources/pending/customer-a/deploy -X PUT
```

The current API doesn't strictly follow the ReST design guidelines and will be updated in a later release.

Step 2 - Update the SNMP configuration

The implementation only supports a *PUT* request because it is an implied "Update" of the configuration since it requires an IP address and all IPs have a default configuration. This request is is passed to the SNMP configuration factory in OpenNMS Horizon for optimization of the configuration store `snmp-config.xml`. This example changes the community string for the IP address 10.1.1.1 to `yRuSonoZ`.

Community string is the only required element

```
curl -v -X PUT -H "Content-Type: application/xml" -H "Accept: application/xml" -d
<snmp-
info><community>yRuSonoZ</community><port>161</port><retries>1</retries><timeout>2000<
/timeout><version>v2c</version></snmp-info>" -u admin:admin
http://localhost:8980/opennms/rest/snmpConfig/10.1.1.1
```

Step 3 - Create/Update the Requisition

This example adds 2 nodes to the Provisioning Group, *customer-a*. Note that the foreign-source attribute typically has a 1 to 1 relationship to the name of the Provisioning Group requisition. There is a direct relationship between the foreign- source attribute in the requisition and the foreign source policy specification. Also, typically, the name of the provisioning group will also be the same. In the following example, the ReST API will automatically create a provisioning group based on the value foreign-source attribute specified in the XML requisition.

```
curl -X POST -H "Content-Type: application/xml" -d "<?xml version="1.0" encoding="UTF-
8"?><model-import xmlns="http://xmlns.opennms.org/xsd/config/model-import" date-
stamp="2009-03-07T17:56:53.123-05:00" last-import="2009-03-07T17:56:53.117-05:00"
foreign-source="customer-a"><node node-label="p-brane" foreign-id="1" ><interface ip-
addr="10.0.1.3" descr="en1" status="1" snmp-primary="P"><monitored-service service-
name="ICMP"/><monitored-service service-name="SNMP"/></interface><category
name="Production"/><category name="Routers"/></node><node node-label="m-brane"
foreign-id="1" ><interface ip-addr="10.0.1.4" descr="en1" status="1" snmp-
primary="P"><monitored-service service-name="ICMP"/><monitored-service service-
name="SNMP"/></interface><category name="Production"/><category
name="Routers"/></node></model-import>" -u admin:admin
http://localhost:8980/opennms/rest/requisitions
```

A provisioning group file called `etc/imports/customer-a.xml` will be found on the OpenNMS Horizon system following the successful completion of this `curl` command and will also be visible via the WebUI.

Add, *Update*, *Delete* operations are handled via the ReST API in the same manner as described in detailed specification.

6.8. Provisioning Single Nodes (Quick Add Node)

Adding a Node to a Current Requisition

Often, it is requested that a single node add/update be completed for an already defined provisioning group. There is a ReST API for the *Add Node* implementation found in the OpenNMS Horizon Web-UI. For this to work, the provisioning group must already exist in the system even if there are no nodes defined in the group.

1. Create a foreign source (if required)
2. Specify SNMP configuration
3. Provide a single node with the following specification

6.9. Fine Grained Provisioning Using *provision.pl*

provision.pl provides an example command-line interface to the provisioning-related OpenNMS Horizon REST API endpoints.

The script has many options but the first 3 optional parameters are described here:

You can use `--help` to the script to see all the available options.

```
--username (default: admin)
--password (default: admin)
--url (default: http://localhost:8980/opennms/rest)
```

6.9.1. Create a new requisition

provision.pl provides easy access to the requisition REST service using the *requisition* option:

```
${OPENNMS_HOME}/bin/provision.pl requisition customer1
```

This command will create a new, empty (containing no nodes) requisition in OpenNMS Horizon.

The new requisition starts life in the `pending` state. This allows you to iteratively build the requisition and then later actually import the nodes in the requisition into OpenNMS Horizon. This handles all adds/changes/deletes at once. So, you could be making changes all day and then at night either have a schedule in OpenNMS Horizon that imports the group automatically or you can send a command through the REST service from an outside system to have the pending requisition imported/reimported.

You can get a list of all existing requisitions with the `list` option of the `provision.pl` script:

```
${OPENNMS_HOME}/bin/provision.pl list
```

Create a new Node

```
${OPENNMS_HOME}/bin/provision.pl node add customer1 1 node-a
```

This command creates a node element in the requisition *customer1* called *node-a* using the script's *node* option. The node's foreign-ID is *1* but it can be any alphanumeric value as long as it is unique within the requisition. Note the node has no interfaces or services yet.

Add an Interface Element to that Node

```
${OPENNMS_HOME}/bin/provision.pl interface add customer1 1 127.0.0.1
```

This command adds an interface element to the node element using the *interface* option to the `provision.pl` command and it can now be seen in the pending requisition by running `provision.pl requisition list customer1`.

Add a Couple of Services to that Interface

```
${OPENNMS_HOME}/bin/provision.pl service add customer1 1 127.0.0.1 ICMP
${OPENNMS_HOME}/bin/provision.pl service add customer1 1 127.0.0.1 SNMP
```

This adds the 2 services to the specified 127.0.0.1 interface and is now in the pending requisition.

Set the Primary SNMP Interface

```
${OPENNMS_HOME}/bin/provision.pl interface set customer1 1 127.0.0.1 snmp-primary P
```

This sets the 127.0.0.1 interface to be the node's Primary SNMP interface.

Add a couple of Node Categories

```
${OPENNMS_HOME}/bin/provision.pl category add customer1 1 Routers
${OPENNMS_HOME}/bin/provision.pl category add customer1 1 Production
```

This adds the two categories to the node and is now in the pending requisition.

These categories are case-sensitive but do not have to be already defined in OpenNMS Horizon. They will be created on the fly during the import if they do not already exist.

Setting Asset Fields on a Node

```
${OPENNMS_HOME}/bin/provision.pl asset add customer1 1 serialnumber 9999
```

This will add value of `9999` to the asset field: *serialnumber*.

Deploy the Import Requisition (Creating the Group)

```
${OPENNMS_HOME}/bin/provision.pl requisition import customer1
```

This will cause OpenNMS Horizon Provisiond to import the pending `customer1` requisition. The formerly pending requisition will move into the `deployed` state inside OpenNMS Horizon.

Deleting a Node from a Requisition

Very much the same as the add, except that a single delete command and a re-import is required. What happens is that the audit phase is run by Provisiond and it will be determined that a node has been removed from the requisition and the node will be deleted from the DB and all services will stop activities related to it.

```
${OPENNMS_HOME}/bin/provision.pl node delete customer1 1 node-a
${OPENNMS_HOME}/bin/provision.pl requisition import customer1
```

This completes the life cycle of managing a node element, iteratively, in a import requisition.

6.10. Yet Other API Examples

List the Nodes in a Provisioning Group

The `provision.pl` script doesn't supply this feature but you can get it via the REST API. Here is an example using `curl`:

```
#!/bin/bash
REQ=$1
curl -X GET -H "Content-Type: application/xml" -u admin:admin
http://localhost:8980/opennms/rest/requisitions/$REQ 2>/dev/null | xmllint --format -
```

6.11. Service Detectors

6.11.1. SNMP Detector

This detector is used to find and assigns services based on *SNMP*. The detector binds a service with a given *Service Name* when a particular *SNMP OID* as scalar or table matches a given criteria.

Detector facts

Implementation	`org.opennms.netmgt.provision.detector.snmp.SnmpDetector`

Configuration and Usage

Table 66. Parameters for the SNMP detector

Parameter	Description	Required	Default value
`oid`	*SNMP OID* for scalar or table to detect the service.	required	`.1.3.6.1.2.1.1.2.0`
`retry`	Number of retries to detect the service.	optional	*agent config*
`timeout`	Timeout in milliseconds to wait for a response from the *SNMP* agent.	optional	*agent config*
`vbvalue`	expected return value to detect the service; if not specified the service is detected if the *SNMP OID* returned any kind of valid value. The `vbvalue` is evaluated as Java Regular Expression.	optional	`-`
`hex`	Set `true` if the data is from type *HEX-String*.	optional	`false`
`isTable`	Set `true` if detector should evaluate *SNMP tables*.	optional	`false`
`matchType`	Set match type to evaluate the expected value in the SNMP table. *EXIST*: the expected `vbalue` is ignored, service detected if the given table under *OID* exist *ALL*: all values in the table must match against expected `vbalue` to detect service *ANY*: at least one value in the table must match against expected `vbalue` to detect service *NONE*: None of the values should match against expected value to detect service	optional	`EXIST`

Example for SNMP scalar value

We have *Dell* server farm and want to monitor the global server status provided by the *OpenManage Server Administrator*. Global status is provided by a scalar *OID* `.1.3.6.1.4.1.674.10892.1.200.10.1.2.1`. The service should be automatically detected if the server supports this *OID*.

For provisioning we have a requisition named *Server* which contains all server of our data center. A *Detector* with the name *Dell-OMSA-Global-State* for this requisition is created with the following parameter:

Table 67. Parameters for the SNMP detector

Parameter	Value
`Name`	`Dell-OMSA-Global-State`
`oid`	`.1.3.6.1.4.1.674.10892.1.200.10.1.2.1`

When the requisition *Server* is synchronized the service *Dell-OMSA-Global-State* will be detected in case they support the given *SNMP OID*.

Example using SNMP tables

We have a *HP* server farm and want to monitor the status of logical drives over *SNMP* provided from *HP Insight Manager*. The status for logical drives is provided in a *SNMP Table* under `.1.3.6.1.4.1.232.3.2.3.1.1.4`. The service should be automatically assigned to all servers exposing the given *SNMP OID*.

For provisioning we have a requisition named *Server* which contains all server of our data center. A *Detector* with the name *HP-Insight-Drive-Logical* for this requisition is created with the following parameter:

Table 68. Parameters for the SNMP detector

Parameter	Value
`Name`	`HP-Insight-Drive-Logical`
`oid`	`.1.3.6.1.4.1.232.3.2.3.1.1.4`
`isTable`	`true`

When the requisition *Server* is synchronized the service *HP-Insight-Drive-Logical* will be detected in case they support the given *SNMP OID* table.

Chapter 7. Business Service Monitoring

This section describes how to model and configure *Business Services (BS)* and orchestrate them in a hierarchy. The concepts and usage of the section *Business Service Monitoring* from the *User Guide* is presumed.

Business Service Monitoring (BSM) includes the following components:

- *Business Service Monitoring Daemon (BSMD)*: Maintains and drives the state of all *BS*
- *Business Service Editor*: Web application which allows you to create, update or delete *BS*
- *Topology View for Business Services*: Visual representation of the *Business Service Hierarchy* as a component of the *Topology User Interface*.
- *BSM ReST API*: ReST based API to create, read, update or delete *BS*

7.1. Business Service Definition

The status of *Service Monitors* and any kind of *Alarm* can be used to drive the *Operational Status* of a *BS*. A *BS* is defined with the following components:

- *Business Service Name*: A unique name used to identify the *BS*
- *Edges*: A set of elements on which this *BS* relies which can include other *BS*, or *Reduction Keys*.
- *Reduce Function*: Function used to aggregate the *Operational Status* from all the *Edges*. Specific functions may take additional parameters.
- *Attributes*: Optional key/value pairs that can be used to tag or enrich the *Busines Service* with additional information.

Each *Business Service* can contain a list of optional key/value attributes. These can be used to identify or tag the *BS*, and may be reference in other workflows. These attributes do not affect the dependencies or the status calculation of the *BS*.

Attributes can be used to filter *BS* in *Ops Board* dashlets.

The *Business Service Editor* is used to manage and model the *Business Services* and their hierarchy. It is required to have administrative permissions and is available in *"Login Name → Configure OpenNMS → Manage Business Services"* in the *Service Monitoring* section.

Managing Business Services with the Business Service Editor

① Create a new *Business Service* definition

② Collapse tree view for all *Business Services* in the view

③ Expand tree view for all *Business Services* in the view

④ Reload all *Business Services* in the view with current *Business Services* from the system

⑤ Reload the *Business Service Monitoring Daemon* to use the *Business Service* definition as configured

⑥ *Business Service* dependency hierarchy as tree view

⑦ Show the current *Business Service* with dependencies in the *Topology UI*

⑧ Edit and delete existing *Business Service* defintions

As shown in figure Managing Business Services with the Business Service Editor the *Business Services* can created or changed. The hierarchy is created by assigning an existing *Business Service* as *Child Service*.

7.2. Edges

Edges map the *Alarm* status monitoring with *OpenNMS*

The following types can be used:

- *Child Service*: A reference to an existing *Business Service* on which to depend
- *IP Service*: A convenient way to refer to the alarms that can be generated by a monitored *IP Service*. This will automatically provided edges for the *nodeLostService*, *interfaceDown* and *nodeDown* reductions keys of the specified service.
- *Reduction Key*: A resolved *Reduction Key* used to refer to a specific *Alarm*, e.g. generated by a *SNMP Trap* or *Threshold* violation

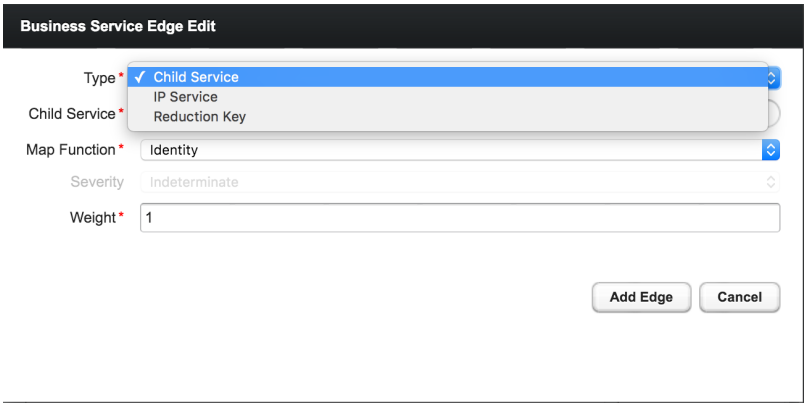
If you need help determining the reduction key used by alarm, trigger the alarm in question and pull the reduction key from the *Alarm* details page.

All edge types have the following parameters:

- *Map Function*: The associated *Map Function* for this *Edge*
- *Weight*: The relative *Weight* of this edge. Used by certain *Reduce Functions*.

Both *IP Service* and *Reduction Key* type edges also support a *Friendly Name* parameter which gives the user control on how the edge is labeled in the *Topology User Interface*. The editor changing the *Edge* attributes is shown in figure Editor to add Business Service Edges.

Editor to add Business Service Edges



7.2.1. Child Services

To create a hierarchy of *Business Services* they need to be created first. The hierarchy is build by selecting the *Business Service* as_Child Service_ as dependency.

7.2.2. IP Services

The *IP Service* is a predefined set of *Reduction Keys* which allows easily to assign a specific *Monitored Service* to the given *BS*. As an example you have multiple Servers with a *Monitored Service SMTP* and you want to model a *BS* named *Mail Communication*. If just the *Reduction Key* for a *nodeLostService* is assgined, the *BS* would not be affected in case the *IP Interface* or the whole *Node* goes down. *OpenNMS* generates *Alarms* with different *UEI* which needs to be assigned to the *BS* as well. To make it easier to model this use case the *IP Service* generates the following *Reduction Keys* automatically:

- `uei.opennms.org/nodes/nodeLostService:%nodeId%:%ipAddress%:%serviceName%`: Matches *Alarms* when the given *Monitored Service* goes down
- `uei.opennms.org/nodes/interfaceDown:%nodeId%:%ipAddress%`: Matches *Alarms* when the given *IP Interface* of the *Monitored Service* goes down
- `uei.opennms.org/nodes/nodeDown:%nodeId%`: Matches *Alarms* when the given *Node* of the *Monitored Service* goes down

7.2.3. Custom Reduction Key

The *Reduction Key* edge is used to refer to specific instance of alarms. When an alarm with the given *Reduction Key* is present, the alarms' severity will be used to calculate the *Operational Status* of the *BS*. To give a better explanation a *Friendly Name* can be set and is used in the *Business Service View*. The format of the *Reduction Key* is build by a set of attributes as a key separated by `:` and enclosed in `%`, i.e (`%attribute%:%attribute%`).

Example of a Reduction Key for a specific nodeLostService

```
%uei.opennms.org/nodes/nodeLostService%:%nodeId%:%ipAddress%:%serviceName%
```

7.3. Map Functions

The *Map Functions* define how the *Severity* of the edge will be used in the *Reduce Function* of the parent when calculating the *Operational Status*.

The available *Map Functions* are:

Table 69. Calculation of the Operational Status with Map Functions

Name	Description
`Identity`	Use the same *Severity* as *Operational Status* of the *BS*
`Increase`	Increase the *Severity* by one level and use it as *Operational Status* of the *BS*
`Decrease`	Decrease the *Severity* by one level and use it as *Operational Status* of the *BS*
`SetTo`	Set the *Operational Status* to a constant *Severity* value
`Ignore`	The input of the *Edge* is ignored for *Operational Status* calcualation

7.4. Reduce Functions

A *Reduce Function* is used to aggregate the *Operational Status* for the *BS*. The *Alarm Severity* from the *Edges* are used as input for the *Reduce Function*. For this operation the following *Reduce Functions* are available:

Table 70. Status calculation Reduce Functions

Name	Description
`Highest Severity`	Uses the value of the highest severity, *Weight* is ignored.
`Threshold`	Uses the highest severity found more often than the given threshold, e.g. 0.26 can also be seen as 26%, which means at least 2 of 4 *Alarms* need to be raised to change the *BS*.
`Highest Severity Above`	Uses the highest severity greater than the given threshold severity.

The following table shows the status calculation with *Edges* assigned to an *IP Service*. The *IP-Service* is driven by the monitoring of the *ICMP* service for three Web Server. In the table below you find a configuration where *Web Server 3* is weighted 3 times higher than the other and a threshold of 0.33 (33%) is configured.

Table 71. Example for status calculation using the Threshold function

Name	Weight	Weight Factor	Input Severity	Operational Status	Critical	Major	Minor	Warning	Normal
Web-ICMP-1	1	0.2	Critical	Critical	0.2	0.2	0.2	0.2	0.2
Web-ICMP-2	1	0.2	Normal	Normal	0	0	0	0	0.2

Name	Weight	Weight Factor	Input Severity	Operational Status	Critical	Major	Minor	Warning	Normal
Web-ICMP-3	3	0.6	Warning	Warning	0	0	0	0.6	0.6
Total		1.0			0.2	0.2	0.2	0.8	1
Percentage		100%			20%	20%	20%	80%	100%

The *Operational Status Severity* is evaluated from left to right, the first value higher then the configured *Threshold* is used. In this case the *Operational Status* is set to *Warning* because the first threshold which exceeds *33%* is *Warning* with *80%*.

7.5. Business Service Daemon

The calculation of the *Operational Status* of the *BS* is driven by the *Business Service Monitoring Daemon* (bsmd). The daemon is responsible for tracking the operational status of all *BS* and for sending events in case of operational status changes. Every time the configuration of a *Business Service* is changed a reload of the daemon's configuration is required. This includes changes like the name of the *Business Service* or its attributes as well as changes regarding the *Reduction Keys*, contained *Business Services* or *IP Services*. The *bsmd* configuration can be reloaded with the following mechanisms:

- Click the *Reload Daemon* button in the *Business Service Editor*
- Send the *reloadDaemonConfig* event using `send-event.pl` or use the WebUI in *Manually Send an Event* with parameter `daemonName bsmd`
- Use the ReST API to perform a `POST` request to `/opennms/api/v2/business-services/daemon/reload`

If the reload of the configuration is done an event of type `uei.opennms.org/internal/reloadDaemonConfigSuccessful` is fired.

Example reloading bsmd configuration from CLI

```
$OPENNMS_HOME/bin/send-event.pl -p 'daemonName bsmd'
uei.opennms.org/internal/reloadDaemonConfig
```

Example reloading bsmd configuration through ReST POST

```
curl -X POST -u admin:admin -v http://localhost:8980/opennms/api/v2/business-
services/daemon/reload
```

Chapter 8. Topology Map

This section describes how to configure the *Topology Map*.

8.1. Icons

Each Vertex on the *Topology Map* is represented by an icon. The default icon is configured in the icon mapping file: `${OPENNMS_HOME}/etc/org.opennms.features.topology.app.icons.<topology-namespace>.cfg`. If an icon mapping file does not exist for a *Topology Provider*, the provider does not support customization.

List of available icon mapping files (may not be complete)

```
org.opennms.features.topology.app.icons.default.cfg ①
org.opennms.features.topology.app.icons.application.cfg ②
org.opennms.features.topology.app.icons.bsm.cfg ③
org.opennms.features.topology.app.icons.linkd.cfg ④
org.opennms.features.topology.app.icons.vmware.cfg ⑤
```

① Default icon mapping

② Icon mapping for the Application Topology Provider

③ Icon mapping for the Business Services Topology Provider

④ Icon mapping for the Linkd Topology Provider

⑤ Icon mapping for the Vmware Topology Provider

Each File contains a mapping in form of `<icon key> = <icon id>`.

Icon key

A *Topology Provider* dependent string which maps to an `icon id`. An `icon key` consists of one to multiple `segments`. Each segment must contain only numbers or characters. If multiple `segments` exist they must be separated by `.`, e.g. `my.custom.key`. Any existing default `icon keys` are not configurable and should not be changed.

Icon id

The `icon id` is a unique icon identifier to reference an icon within one of the available SVG icons located in `${OPENNMS_HOME}/jetty-webapps/opennms/svg`. For more details see Add new icons.

Icon key and icon id specification using BNF

```
icon key ::= segment["."segment]*
segment ::= text+ [( "-" | "_" | ":" ) text]*
text ::== (char | number)+
char ::== A | B | ... | Z | a | b | ... | z
number ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
icon id ::= segment
```

Example icon mapping file

```
# Business Service Topology
bsm.business-service = business_service ①
bsm.ip-service = IP_service ②
bsm.reduction-key = reduction_key ③
```

① Icon definition for Business Services

② Icon definition for IP Services

③ Icon definition for Reduction Keys

8.1.1. Icon resolution

The icon of a vertex is resolved as follows:

- If a `vertex id` to `icon id` mapping is defined, the icon referenced by the `icon id` is used
- If a mapping for the `icon key` determined by the *Topology Provider* for the vertex is defined, the icon referenced by the `icon id` is used
  - If no mapping exists and the `icon key` has more than one `segments`, reduce the `icon key` by the last `segment` and try resolving that `icon key`
- If no mapping is defined, the fallback `icon key default` is used.

The following example icon mapping is defined for the *Linkd Topology Provider* to illustrate this behaviour.

```
linkd.system.snmp.1.3.6.1.4.1.9.1.485 = server1
linkd.system.snmp.1.3.6 = server2
```

If the Enterprise OID of a node is `1.3.6.1.4.1.9.1.485` the icon with id `server1` is used. If the Enterprise OID of a node is `1.3.6` the icon with id `server2` is used. However, if the Enterprise OID of a node is `1.3.6.1.4.1.9.1.13` the icon with id `server2` is used.

Linkd Topology Provider

The *Linkd Topology Provider* uses the `Enterprise OID` from each node to determine the icon of a vertex.

8.1.2. Change existing icon mappings

The easiest way to change an icon representation of an existing Vertex is to use the *Icon Selection Dialog* from the Vertex' context menu in the *Topology Map*. This will create a custom `icon key` to `icon id` mapping in the *Topology Provider* specific icon mapping file. As `icon key` the Vertex id is used. This allows each Vertex to have it's own icon.

If a more generic approach is preferred the icon mapping file can be modified manually.

Do NOT remove the default mappings and do NOT change the icon keys in the default mappings.

8.1.3. Add new icons

All available icons are stored in SVG files located in `${OPENNMS_HOME}/jetty-webapps/opennms/svg`. To add new icons, either add definitions to an existing SVG file or create a new SVG file in that directory.

Whatever way new icons are added to *OpenNMS* it is important that each new `icon id` describes a set of icons, rather than a single icon. The following example illustrates this.

Example SVG file with a custom icon with id `my-custom`

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg id="icons" xmlns="http://www.w3.org/2000/svg">
  <g id="my-custom_icon"> ①
      <g id="my-custom_active"> ②
          <!-- rect, path, circle, etc elements, supported by SVG -->
      </g>
      <g id="my-custom_rollover"> ③
          <!-- rect, path, circle, etc elements, supported by SVG -->
      </g>
      <g id="my-custom"> ④
          <!-- rect, path, circle, etc elements, supported by SVG -->
      </g>
  </g>
  <!-- Additional groups ... -->
</svg>
```

① Each icon must be in a SVG group with the id `<icon id>_icon`. Each SVG `<icon id>_icon` group must contain three sub groups with the ids: `<icon id>_active`, `<icon id>_rollover` and `<icon id>`.

② The icon to use when the Vertex is selected.

③ The icon to use when the Vertex is moused over.

④ The icon to use when the Vertex is not selected or not moused over (just visible).

It is important that each `icon id` is unique overall SVG files. This means there cannot be another `my-custom` icon id in any other SVG file.

If the new icons should be selectable from the *Topology Map's Icon Selection Dialog* an entry with the new `icon id` must be added to the file `${OPENNMS_HOME}/etc/org.opennms.features.topology.app.icons.properties`.

Snippet of `org.opennms.features.topology.app.icons.list`

```
access_gateway ①
accesspoint
cloud
fileserver
linux_file_server
opennms_server
printer
router
workgroup_switch
my-custom ②
```

① Already existing icon ids

② New icon id

The order of the entries in `org.opennms.features.topology.app.icons.list` determine the order in the *Icon Selection Dialog* in the *Topology Map*.

Chapter 9. Database Reports

Reporting on information from the *OpenNMS Horizon* monitoring system is important for strategical or operational decisions. *Database Reports* give access to the embedded *JasperReports* engine and allows to create and customize report templates. These reports can be executed on demand or on a pre-defined schedule within *OpenNMS Horizon*.

Originally *Database Reports* were introduced to create reports working on data stored in the *OpenNMS Horizon* database only. This is no longer mandatory, also performance data can be used. Theoretically the reports do not necessarily need to be *OpenNMS Horizon* related.

The *OpenNMS Horizon Report Engine* allows the creation of various kinds of reports and also supports distributed report repositories. At the moment these features are not covered by this documentation. Only reports using *JasperReports* are described here.

9.1. Overview

The *OpenNMS Horizon Report Engine* uses the *JasperReport* library to create reports in various output formats. Each report template must be a `*.jrxml` file. The *OpenNMS Horizon Report Engine* passes a *JDBC* Connection to the *OpenNMS Horizon Database* to each report on execution.

Table 72. feature overview

Supported Output Formats	`PDF`, `CSV`
`JasperReport` Version	6.1.1

For more details on how *JasperReports* works, please have a look at the official documentation of *Jaspersoft Studio*.

9.2. Add a custom report

To add a new *JasperReport* report to the *Local OpenNMS Horizon Report Repository*, the following steps are required.

At first a new entry in the file `$OPENNMS_HOME/etc/database-reports.xml` must be created.

```
<report
  id="MyReport" ①
  display-name="My Report" ②
  online="true" ③
  report-service="jasperReportService" ④
  description="This is an example description. It shows up in the web ui when creating
an online report"  ⑤
/>
```

① A unique identifier.

② The name of the report. Is shown when using the web ui.

③ Defines if this report can be executed on demand, otherwise only scheduling is possible.

④ The report service implementation to use. In most cases this is `jasperReportService`.

⑤ A description of the report. Is shown when using the web ui.

In addition a new entry in the file `$OPENNMS_HOME/etc/jasper-reports.xml` must be created.

```
<report
  id="MyReport" ①
  template="My-Report.jrxml" ②
  engine="jdbc" ③
/>
```

① The identifier defined in the previous step. This identifier must exist in `$OPENNMS_HOME/etc/database-reports.xml`.

② The name of the template. The template must be located in `$OPENNMS_HOME/etc/report-templates`.

③ The engine to use. It is either `jdbc` or `null`.

9.3. Use of Jaspersoft Studio

When developing new reports it is recommended to use the *Jaspersoft Studio* application. It can be downloaded here.

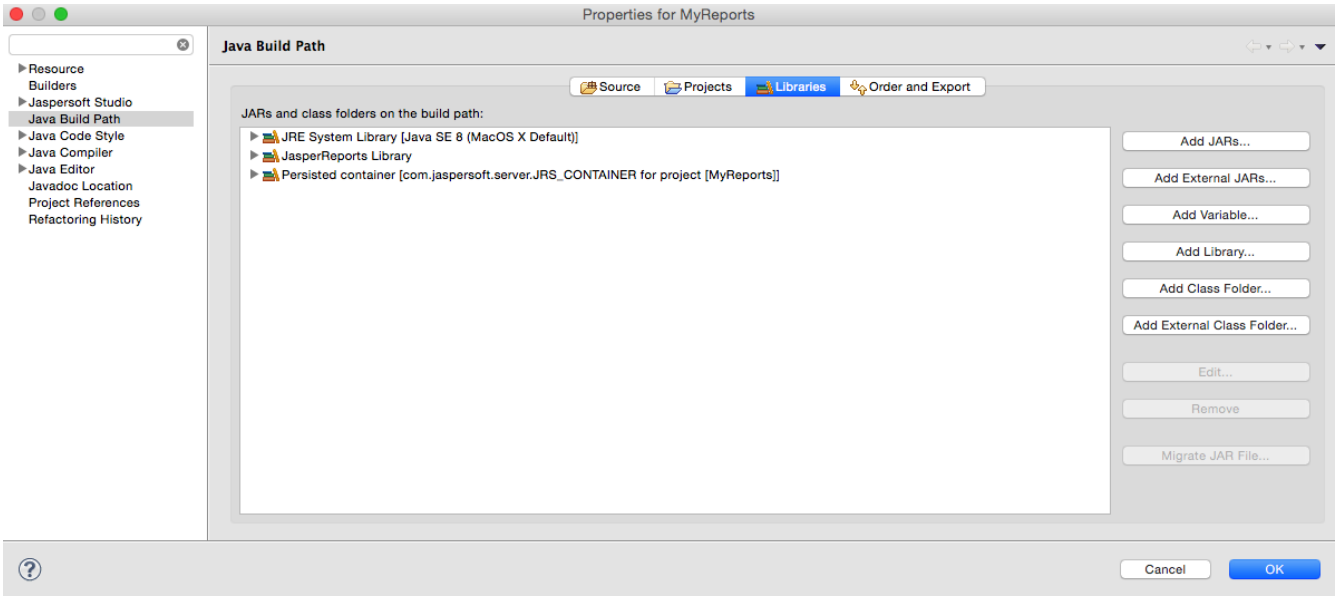
We recommend always to use the same *Jaspersoft Studio* version as the *JasperReport* library OpenNMS Horizon uses. Currently OpenNMS Horizon uses version 6.1.1.

9.3.1. Connect to the OpenNMS Horizon Database

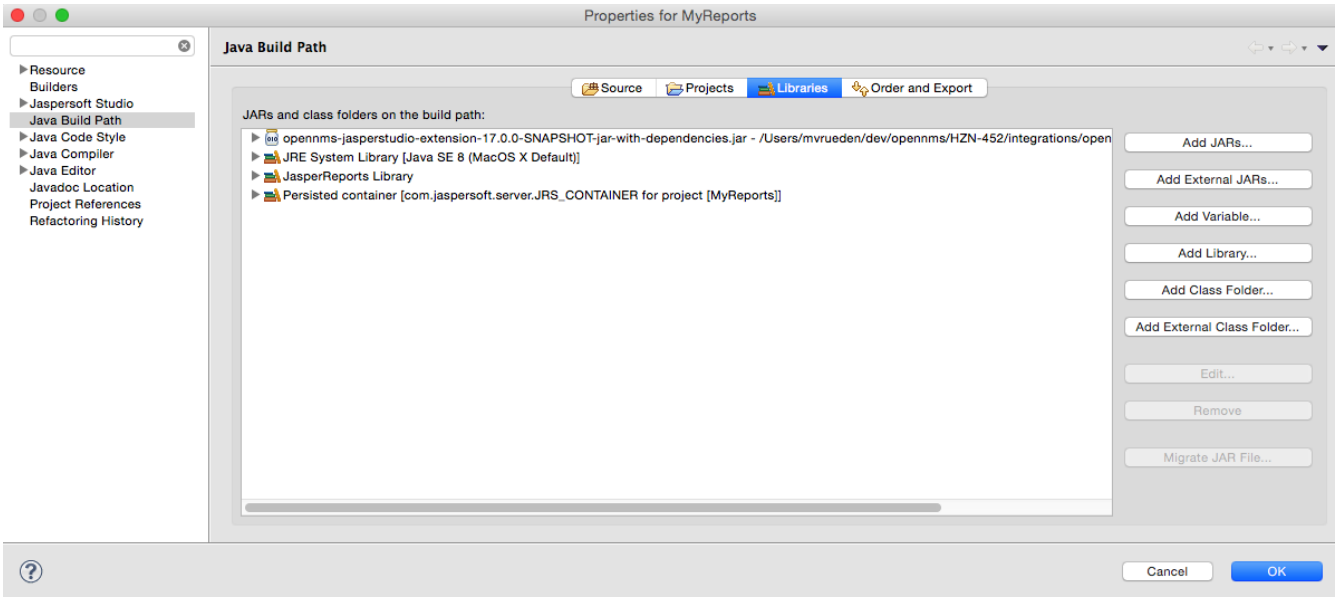
In order to actually create SQL statements against the *OpenNMS Horizon database* a `database Data Adapter` must be created. The official *Jaspersoft Studio* documentation and wiki covers this aspect.

9.3.2. Use Measurements Datasource and Helpers

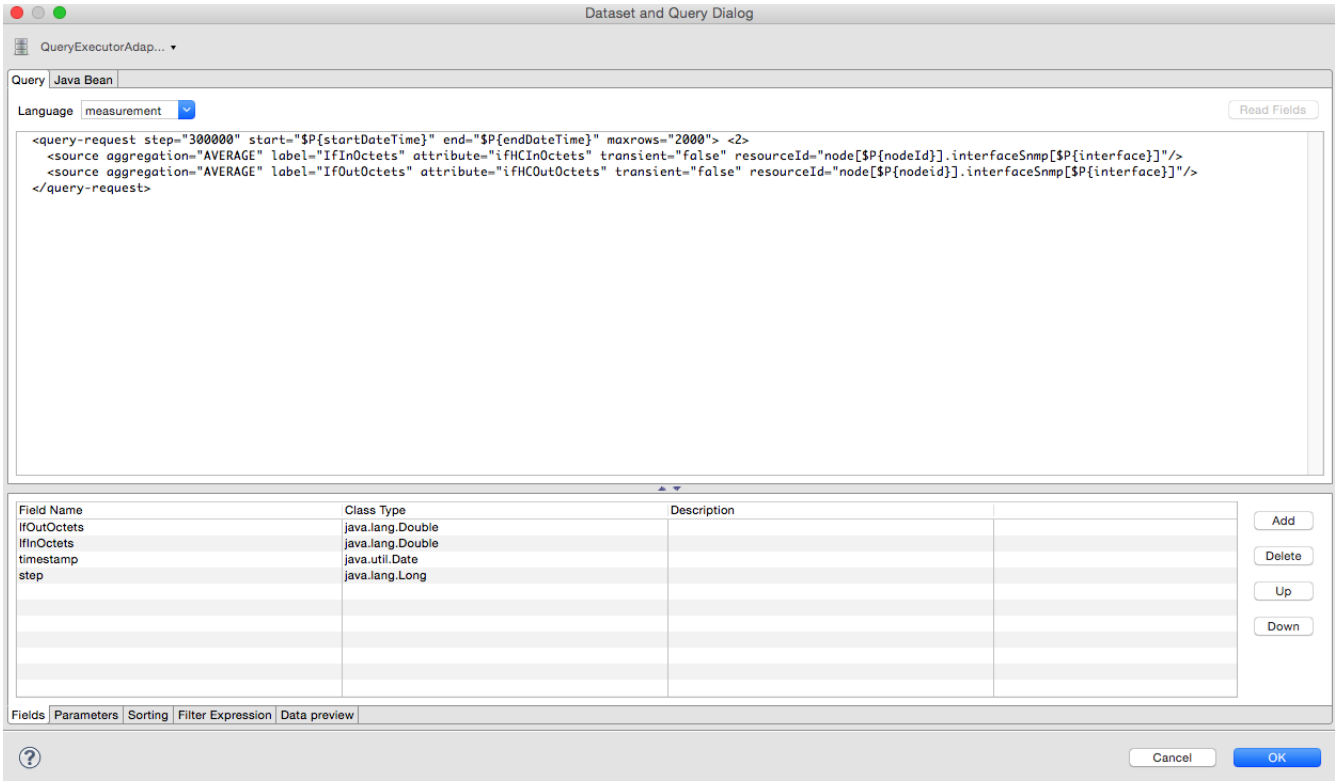
To use the *Measurements API* it is required to add the *Measurements Datasource* library to the build path of *JasperStudio*. This is achieved with right click in the `Project Explorer` and select `Configure Buildpath`.



1. Switch to the `Libraries` tab.
2. Click `Add External JARs` and select the `opennms-jasperstudio-extension-18.0.0-jar-with-dependencies.jar` file located in `$OPENNMS_HOME/contrib/jasperstudio-extension`.
3. Close the file selection dialog.



1. Close the dialog.
2. The *Measurements Datasource and Helpers* should now be available.
3. Go to the `Dataset and Query Dialog` in *Jaspersoft Studio* and select a language called `measurement`.



Even if there is no `Read Fields` functionality available, the `Data preview` can be used. It is required the the access to the *Measurements API* is possible using the connection parameters `MEASUREMENT_URL`, `MEASUREMENT_USERNAME` and `MEASUREMENT_PASSWORD`. The Supported Fields section gives more details. In addition you have

9.4. Accessing Performance Data

Before *OpenNMS Horizon 17* and *OpenNMS Meridian 2016* it was possible to access the performance data stored in `.rrd` or `.jrobin` files directly by using the `jrobin` language extension provided by the `RrdDataSource`. This is no longer possible and the *Measurements Datasource* has to be used.

To access performance data within reports we created a custom *Measurement Datasource* which allows to query the *Measurements API* and process the returned data in your reports. Please refer to the official Measurements API documentation on how to use the _Measurements API_.

When using the *Measurements Datasource* within a report a `HTTP` connection to the *Measurements API* is only established if the report is NOT running within OpenNMS Horizon, e.g. when used with *Jaspersoft Studio*.

To receive data from the *Measurements API* simply create a query as follows:

Sample queryString to receive data from the `Measurements API`

```
<query-request step="300000" start="$P{startDateTime}" end="$P{endDateTime}" maxrows=
"2000"> ①
  <source aggregation="AVERAGE" label="IfInOctets" attribute="ifHCInOctets" transient
="false" resourceId="node[$P{nodeId}].interfaceSnmp[$P{interface}]"/>
  <source aggregation="AVERAGE" label="IfOutOctets" attribute="ifHCOutOctets"
transient="false" resourceId="node[$P{nodeid}].interfaceSnmp[$P{interface}]"/>
</query-request>
```

① The query language. In our case `measurement`, but `JasperReports` supports a lot out of the box, such as `sql`, `xpath`, etc.

9.4.1. Fields

Each datasource should return a number of fields, which then can be used in the report. The *Measurement Datasource* supports the following fields:

Field name	Field type	Field description
`<label>`	`java.lang.Double`	Each `Source` defined as `transient=false` can be used as a field. The name of the field is the `label`, e.g. `IfInOctets`
`timestamp`	`java.util.Date`	The timestamp of the sample.
`step`	`java.lang.Long`	The `Step` size of the `Response`. Returns the same value for all rows.
`start`	`java.lang.Long`	The `Start` timestamp in milliseconds of the `Resopnse`. Returns the same value for all rows.
`end`	`java.lang.Long`	The `End` timestamp in milliseconds of the `Response`. Returns the same value for all rows.

For more details about the `Response`, please refer to the official Measurement API documentation.

9.4.2. Parameters

In addition to the `queryString` the following *JasperReports* parameters are supported.

Parameter name	Required	Description
MEASUREMENT_URL	`yes`	The URL of the *Measurements API*, e.g. `http://localhost:8980/opennms/rest/measurements`
MEASUREMENT_USERNAME	`no`	If authentication is required, specify the username, e.g. `admin`
MEASUREMENT_PASSWORD	`no`	If authentication is required, specify the password, e.g. `admin`

9.5. Helper methods

There are a couple of helper methods to help creating reports in *OpenNMS Horizon*.

These helpers come along with the *Measurement Datasource*.

Table 73. supported helper methods

Helper class	Helper Method	Description
`org.opennms.netmgt.jasper.helper.MeasurementsHelper`	`getNodeOrNodeSourceDescriptor(nodeId, foreignSource, foreignId)`	Generates a `node source descriptor` according to the input paramters. Either `node[nodeId]` or `nodeSource[foreignSource:foreignId]` is returned. `nodeSource[foreignSource:foreignId]` is only returned if `foreignSource` and `foreignId` is not empty and not null. Otherwise always `node[nodeId]` is returned. `nodeId` : String, the id of the node `foreignSource`: String, the foreign source of the node, may be null `foreignId`: String, the foreign id of the node, may be null. For more details checkout Usage of the node source descriptor.

Helper class	Helper Method	Description
`org.opennms.netmgt.jasper.helper.MeasurementsHelper`	`getInterfaceDescriptor(snmpifname, snmpifdescr, snmphysaddr)`	Returns the `interface descriptor` of a given interface, e.g. `en0-005e607e9e00`. The input paramaters are prioritized. If a `snmpifdescr` is specified, it is used instead of the `snmpifname`. It a `snmpifdescr` is defined, it will be appended to `snmpifname` /`snmpifdescr`. `snmpifname`: String, the interface name of the interface, e.g. `en0`. May be null. `snmpifdescr`: String, the description of the interface, e.g. `en0`. May be null. `snmphyaddr`: String, the mac address of the interface, e.g. `005e607e9e00`. May be null. As each input parameter may be null, not all of them can be null at the same time. At least one input parameter has to be defined. For more details checkout Usage of the interface descriptor.

9.5.1. Usage of the interface descriptor

An `interfaceSnmp` is addressed with the exact `interface descriptor`. To allow easy access to the `interface descriptor` a helper tool is provided. The following example shows the usage of that helper.

jrxml report snippet to visualize the use of the `interface descriptor`

```
<parameter name="interface" class="java.lang.String" isForPrompting="false">
  <parameterDescription><![CDATA[]]></parameterDescription>
  <defaultValueExpression>
<![CDATA[org.opennms.netmgt.jasper.helper.MeasurementsHelper.getInterfaceDescriptor($P
{snmpifname}, $P{snmpifdescr}, $P{snmpphysaddr})]]></defaultValueExpression>
</parameter>
<queryString language="Measurement">
  <![CDATA[<query-request step="300000" start="$P{startDateTime}"
end="$P{endDateTime}" maxrows="2000">
<source aggregation="AVERAGE" label="IfInOctets" attribute="ifHCInOctets"
transient="false" resourceId="node[$P{nodeId}].interfaceSnmp[$P{interface}]"/>
<source aggregation="AVERAGE" label="IfOutOctets" attribute="ifHCOutOctets"
transient="false" resourceId="node[$P{nodeId}].interfaceSnmp[$P{interface}]"/>
</query-request>]]>
```

9.5.2. Usage of the node source descriptor

A node is addressed by a `node source descriptor`. The `node source descriptor` references the node either via the `foreign source` and `foreign id` or by the `node id`.

If `store by foreign source` is enabled only addressing the node via `foreign source` and `foreign id` is possible.

In order to make report creation easier, there is a helper method to create the `node source descriptor`.

For more information about `store by foreign source`, please have a look at our Wiki.

The following example shows the usage of that helper.

jrxml report snippet to visualize the use of the `node source descriptor`.

```
<parameter name="nodeResourceDescriptor" class="java.lang.String" isForPrompting=
"false">
  <defaultValueExpression>
<![CDATA[org.opennms.netmgt.jasper.helper.MeasurementsHelper.getNodeOrNodeSourceDescri
ptor(String.valueOf($P{nodeid}), $P{foreignsource}, $P{foreignid})
]]></defaultValueExpression>
</parameter>
<queryString language="Measurement">
  <![CDATA[<query-request step="300000" start="$P{startDateTime}"
end="$P{endDateTime}" maxrows="2000">
<source aggregation="AVERAGE" label="IfInOctets" attribute="ifHCInOctets"
transient="false" resourceId="$P{nodeResourceDescriptor}.interfaceSnmp[en0-
005e607e9e00]"/>
<source aggregation="AVERAGE" label="IfOutOctets" attribute="ifHCOutOctets"
transient="false" resourceId="$P{nodeResourceDescriptor}.interfaceSnmp[en0-
005e607e9e00]"/>
</query-request>]]>
```

Depending on the input parameters you either get a `node resource descriptor` or a `foreign source/foreign id resource descriptor`.

9.5.3. Usage of the interface descriptor

An `interfaceSnmp` is addressed with the exact `interface descriptor`. To allow easy access to the `interface descriptor` a helper tool is provided. The following example shows the usage of that helper.

jrxml report snippet to visualize the use of the `interface descriptor`

```
<parameter name="interface" class="java.lang.String" isForPrompting="false">
  <parameterDescription><![CDATA[]]></parameterDescription>
  <defaultValueExpression>
<![CDATA[org.opennms.netmgt.jasper.helper.MeasurementsHelper.getInterfaceDescriptor($P
{snmpifname}, $P{snmpifdescr}, $P{snmpphysaddr})]]></defaultValueExpression>
</parameter>
<queryString language="Measurement">
  <![CDATA[<query-request step="300000" start="$P{startDateTime}"
end="$P{endDateTime}" maxrows="2000">
<source aggregation="AVERAGE" label="IfInOctets" attribute="ifHCInOctets"
transient="false" resourceId="node[$P{nodeId}].interfaceSnmp[$P{interface}]"/>
<source aggregation="AVERAGE" label="IfOutOctets" attribute="ifHCOutOctets"
transient="false" resourceId="node[$P{nodeId}].interfaceSnmp[$P{interface}]"/>
</query-request>]]>
```

To get the appropriate `interface descriptor` depends on the input parameter.

9.5.4. Use HTTPS

To establish a secure connection to the *Measurements API* the public certificate of the running *OpenNMS Horizon* must be imported to the *Java Trust Store*. In Addition *OpenNMS Horizon* must be configured to use that *Java Trust Store*. Please follow the instructions in this chapter to setup the *Java Trust Store* correctly.

In addition please also set the property `org.opennms.netmgt.jasper.measurement.ssl.enable` in `$OPENNMS_HOME\etc\opennms.properties` to `true` to ensure that only secure connections are established.

If `org.opennms.netmgt.jasper.measurement.ssl.enable` is set to `false` an accidentally insecure connection can be established to the *Measurements API* location. A SSL secured connection can be established even if `org.opennms.netmgt.jasper.measurement.ssl.enable` is set to `false`.

9.6. Limitations

- Only a *JDBC Datasource* to the *OpenNMS Horizon Database connection* can be passed to a report, or no datasource at all. One does not have to use the datasource, though.

Chapter 10. Enhanced Linkd

Enhanced Linkd (Enlinkd) has been designed to discover connections between nodes using data generated by various link discovery protocols and accessible via SNMP. *Enlinkd* gathers this data on a regular interval and creates a snapshot of a device's neighbors from its perspective. The connections discovered by *Enlinkd* are called *Links*. The term *Link*, within the context of *Enlinkd*, is not synonymous with the term "link" when used with respect to the network OSI *Layer 2* domain, whereby a link only indicates a *Layer 2* connection. A *Link* in context of *Enlinkd* is a more abstract concept and is used to describe any connection between two *OpenNMS Horizon Nodes*. These *Links* are discovered based on information provided by an agent's understanding of connections at the OSI *Layer 2*, *Layer 3*, or other OSI layers.

The following sections describe the *Enlinkd* daemon and its configuration. Additionally, the supported *Link discovery* implementations will be described as well as a list of the SNMP MIBs that the SNMP agents must expose in order for *EnLinkd* to gather *Links* between *Nodes*. FYI: Detailed information about a node's connections (discovered *Links*) and supporting link data can be seen on the *Node detail page* within the *OpenNMS Horizon* Web-UI.

10.1. Enlinkd Daemon

Essentially *Enlinkd* asks each device the following question: "What is the network topology from your point of view". From this point of view this will only provide local topology discovery features. It does not attempt to discover global topology or to do any correlation with the data coming from other nodes.

For large environments the behavior of *Enlinkd* can be configured. During the *Link* discovery process informational and error output is logged to a global log file.

Table 74. Global log and configuration files for Enlinkd

File	Location	Description
`enlinkd-configuration.xml`	`$OPENNMS_HOME/etc`	Global configuration for the daemon process
`enlinkd.log`	`$OPENNMS_HOME/logs`	Global *Enlinkd* log file
`log4j2.xml`	`$OPENNMS_HOME/etc`	Configuration file to set the log level for *Enlinkd*

Configuration file for Enlinkd

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<enlinkd-configuration threads="5"
                       initial_sleep_time="60000"
                       rescan_interval="86400000"
                       use-cdp-discovery="true"
                       use-bridge-discovery="true"
                       use-lldp-discovery="true"
                       use-ospf-discovery="true"
                       use-isis-discovery="true"
                       />
```

Table 75. Descriptione for global configuration parameter

Attribute	Type	Default	Description
threads	*Integer*	5	Number of parallel threads used to discover the topology.
initial_sleep_time	*Integer*	60000	Time in milliseconds to wait for discovering the topology after OpenNMS Horizon is started.
rescan_interval	*Integer*	86400000	Interval to rediscover and update the topology in milliseconds.
use-cdp-discovery	*Boolean*	true	Enable or disable topology discovery based on *CDP* information.
use-bridge-discovery	*Boolean*	true	Enable or disable algorithm to discover the topology based on the *Bridge MIB* information.
use-lldp-discovery	*Boolean*	true	Enable or disable topology discovery based on *LLDP* information.
use-ospf-discovery	*Boolean*	true	Enable or disable topology discovery based on *OSPF* information.
use-isis-discovery	*Boolean*	true	Enable or disable topology discovery based on *IS-IS* information.

If multiple protocols are enabled, the links will be discovered for each enabled discovery protocol. The topology WebUI will visualize *Links* for each discovery protocol. For example if you start *CDP* and *LLDP* discovery, the WebUI will visualize a *CDP Link* and an *LLDP Link*.

10.2. Layer 2 Link Discovery

Enlinkd is able to discover *Layer 2* network links based on the following protocols:

- Link Layer Discovery Protocol (LLDP)

- Cisco Discovery Protocol (CDP)
- Transparent Bridge Discovery

This information are provided by *SNMP Agents* with appropriate *MIB support*. For this reason it is required to have a working *SNMP* configuration running. The following section describes the required *SNMP MIB* provided by the *SNMP agent* to allow the *Link Discovery*.

10.2.1. LLDP Discovery

The *Link Layer Discovery Protocol (LLDP)* is a vendor-neutral link layer protocol. It is used by network devices for advertising their identity, capabilities, and neighbors. *LLDP* performs functions similar to several proprietary protocols, such as the *Cisco Discovery Protocol (CDP)*, *Extreme Discovery Protocol*, *Foundry Discovery Protocol (FDP)*, *Nortel Discovery Protocol (also known as SONMP)*, and *Microsoft's Link Layer Topology Discovery (LLTD)* *[1: Wikipedia LLDP: https://en.wikipedia.org/wiki/Link_Layer_Discovery_Protocol]*.

Only nodes with a running *LLDP* process can be part of the link discovery. The data is similar to running a `show lldp neighbor` command on the device. Linux and Windows servers don't have an *LLDP* process running by default and will not be part of the link discovery.

The following OIDs are supported to discover and build the *LLDP* network topology.

Table 76. Supported OIDs from LLDP-MIB

Name	OID	Description
lldpLocChassisIdSubtype	`.1.0.8802.1.1.2.1.3.1.0`	The type of encoding used to identify the chassis associated with the local system. Possible values can be: *chassisComponent(1)* *interfaceAlias(2)* *portComponent(3)* *macAddress(4)* *networkAddress(5)* *interfaceName(6)* *local(7)*
lldpLocChassisId	`.1.0.8802.1.1.2.1.3.2.0`	The string value used to identify the chassis component associated with the local system.
lldpLocSysName	`.1.0.8802.1.1.2.1.3.3.0`	The string value used to identify the system name of the local system. If the local agent supports IETF RFC 3418, *lldpLocSysName* object should have the same value of *sysName* object.
lldpLocPortIdSubtype	`.1.0.8802.1.1.2.1.3.7.1.2`	The type of port identifier encoding used in the associated *lldpLocPortId* object.
lldpLocPortId	`.1.0.8802.1.1.2.1.3.7.1.3`	The string value used to identify the port component associated with a given port in the local system.

Name	OID	Description
lldpLocPortDesc	.1.0.8802.1.1.2.1.3.7.1.4	The string value used to identify the *802 LAN station's* port description associated with the local system. If the local agent supports *IETF RFC 2863*, *lldpLocPortDesc* object should have the same value of *ifDescr* object.
lldpRemChassisIdSubtype	.1.0.8802.1.1.2.1.4.1.1.4	The type of encoding used to identify the chassis associated with the local system. Possible values can be: *chassisComponent(1)* *interfaceAlias(2)* *portComponent(3)* *macAddress(4)* *networkAddress(5)* *interfaceName(6)* *local(7)*
lldpRemChassisId	.1.0.8802.1.1.2.1.4.1.1.5	The string value used to identify the chassis component associated with the remote system.

Name	OID	Description
lldpRemPortIdSubtype	.1.0.8802.1.1.2.1.4.1.1.6	The type of port identifier encoding used in the associated *lldpRemPortId* object. *interfaceAlias(1)* the octet string identifies a particular instance of the *ifAlias* object (defined in IETF RFC 2863). If the particular *ifAlias* object does not contain any values, another port identifier type should be used. *portComponent(2)* the octet string identifies a particular instance of the *entPhysicalAlias* object (defined in IETF RFC 2737) for a port or backplane component. *macAddress(3)* this string identifies a particular unicast source address (encoded in network byte order and IEEE 802.3 canonical bit order) associated with the port (IEEE Std 802-2001). *networkAddress(4)* this string identifies a network address associated with the port. The first octet contains the *IANA AddressFamilyNumbers* enumeration value for the specific address type, and octets 2 through N contain the *networkAddress* address value in network byte order. *interfaceName(5)* the octet string identifies a particular instance of the *ifName* object (defined in IETF RFC 2863). If the particular *ifName* object does not contain any values, another port identifier type should be used. *agentCircuitId(6)* this string identifies a agent-local identifier of the circuit (defined in RFC 3046) *local(7)* this string identifies a locally assigned port ID.
lldpRemPortId	.1.0.8802.1.1.2.1.4.1.1.7	The string value used to identify the port component associated with the remote system.
lldpRemPortDesc	.1.0.8802.1.1.2.1.4.1.1.8	The string value used to identify the description of the given port associated with the remote system.
lldpRemSysName	.1.0.8802.1.1.2.1.4.1.1.9	The string value used to identify the system name of the remote system.

Generic information about the *LLDP* process can be found in the *LLDP Information* box on the *Node Detail Page* of the device. Information gathered from these OIDs will be stored in the following database table:

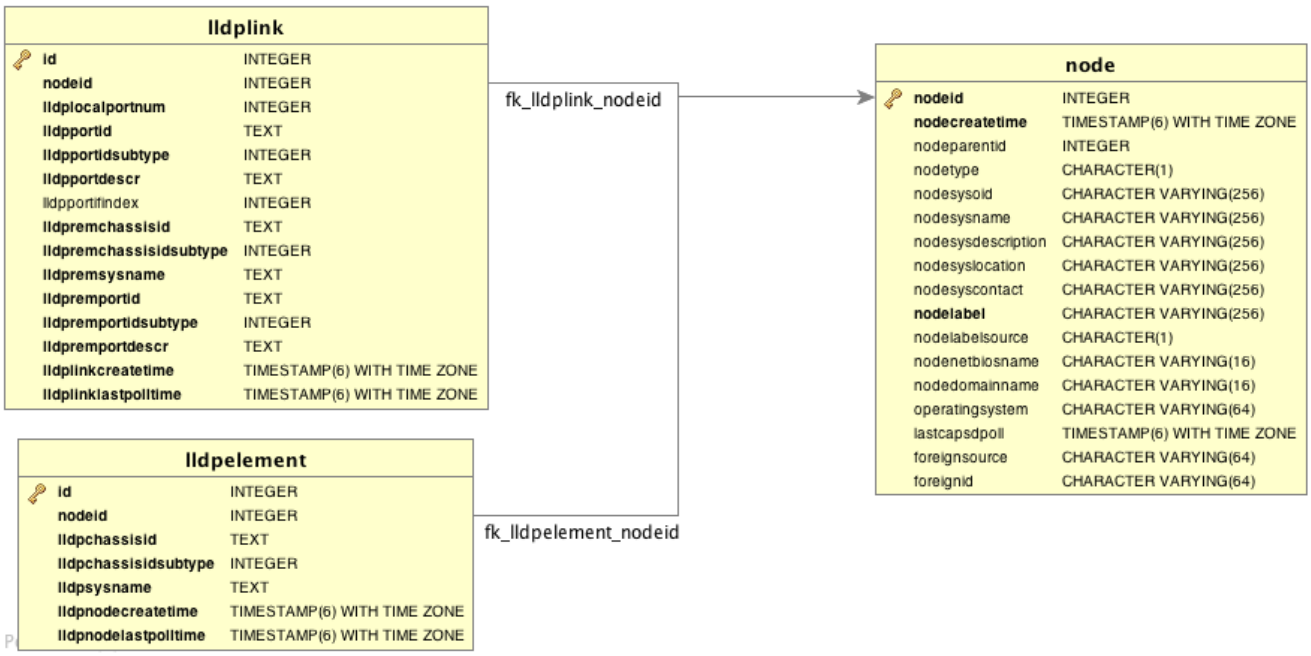


Figure 17. Database tables related to LLDP discovery

10.2.2. CDP Discovery

The *Cisco Discovery Protocol (CDP)* is a proprietary link layer protocol from *Cisco*. It is used by network devices to advertise identity, capabilities and neighbors. *CDP* performs functions similar to several proprietary protocols, such as the *Link Layer Discovery Protocol (LLDP)*, *Extreme Discovery Protocol*, *Foundry Discovery Protocol (FDP)*, *Nortel Discovery Protocol (also known as SONMP)*, and *Microsoft's Link Layer Topology Discovery (LLTD)*. The *CDP* discovery uses information provided by the CISCO-CDP-MIB and CISCO-VTP-MIB.

Only nodes with a running *CDP* process can be part of the link discovery. The data is similar to running a `show cdp neighbor` command on the IOS CLI of the device. Linux and Windows servers don't have a *CDP* process running by default and will not be part of the link discovery.

The following OIDs are supported to discover and build the *CDP* network topology.

Table 77. Supported OIDS from the IF-MIB

Name	OID	Description
ifDescr	`.1.3.6.1.2.1.2.2.1.2`	A textual string containing information about the interface. This string should include the name of the manufacturer, the product name and the version of the interface hardware/software.

Table 78. Supported OIDS from the CISCO-CDP-MIB to discover links

Name	OID	Description
cdpInterfaceName	`.1.3.6.1.4.1.9.9.23.1.1.1.1.6`	The name of the local interface as advertised by *CDP* in the *Port-ID TLV*.

Name	OID	Description
cdpCacheEntry	.1.3.6.1.4.1.9.9.23.1.2.1.1	An entry (conceptual row) in the *cdpCacheTable*, containing the information received via *CDP* on one interface from one device. Entries appear when a *CDP* advertisement is received from a neighbor device. Entries disappear when *CDP* is disabled on the interface, or globally.
cdpCacheAddressType	.1.3.6.1.4.1.9.9.23.1.2.1.1.3	An indication of the type of address contained in the corresponding instance of *cdpCacheAddress*.
cdpCacheAddress	.1.3.6.1.4.1.9.9.23.1.2.1.1.4	The (first) network-layer address of the device's SNMP-agent as reported in the Address *TLV* of the most recently received *CDP* message. For example, if the corresponding instance of *cacheAddressType* had the value `ip(1)`, then this object would be an IP-address.
cdpCacheVersion	.1.3.6.1.4.1.9.9.23.1.2.1.1.5	The Version string as reported in the most recent *CDP* message. The zero-length string indicates no Version field *(TLV)* was reported in the most recent *CDP* message.
cdpCacheDeviceId	.1.3.6.1.4.1.9.9.23.1.2.1.1.6	The *Device-ID* string as reported in the most recent *CDP* message. The zero-length string indicates no *Device-ID* field *(TLV)* was reported in the most recent *CDP* message.
cdpCacheDevicePort	.1.3.6.1.4.1.9.9.23.1.2.1.1.7	The *Port-ID* string as reported in the most recent *CDP* message. This will typically be the value of the *ifName* object (e.g., `Ethernet0`). The zero-length string indicates no *Port-ID* field *(TLV)* was reported in the most recent *CDP* message.
cdpCachePlatform	.1.3.6.1.4.1.9.9.23.1.2.1.1.8	The Device's Hardware Platform as reported in the most recent *CDP* message. The zero-length string indicates that no Platform field *(TLV)* was reported in the most recent *CDP* message.
cdpGlobalRun	.1.3.6.1.4.1.9.9.23.1.3.1.0	An indication of whether the Cisco Discovery Protocol is currently running. Entries in *cdpCacheTable* are deleted when CDP is disabled.
cdpGlobalDeviceId	.1.3.6.1.4.1.9.9.23.1.3.4.0	The device ID advertised by this device. The format of this device id is characterized by the value of *cdpGlobalDeviceIdFormat* object.

Name	OID	Description
cdpGlobalDeviceIdFormat	.1.3.6.1.4.1.9.9.23.1.3.7.0	An indication of the format of Device-Id contained in the corresponding instance of *cdpGlobalDeviceId*. User can only specify the formats that the device is capable of as denoted in *cdpGlobalDeviceIdFormatCpb* object. ***serialNumber(1):*** indicates that the value of *cdpGlobalDeviceId* object is in the form of an *ASCII* string contain the device serial number. ***macAddress(2):*** indicates that the value of *cdpGlobalDeviceId* object is in the form of Layer 2 MAC address. ***other(3):*** indicates that the value of *cdpGlobalDeviceId* object is in the form of a platform specific *ASCII* string contain info that identifies the device. For example: *ASCII string* contains *serialNumber* appended/prepened with system name.

Table 79. Supported OIDS from the CISCO-VTP-MIB.

vtpVersion	.1.3.6.1.4.1.9.9.46.1.1.1.0	**The version of VTP in use on the local system. A device will report its version capability and not any particular version in use on the device. If the device does not support *VTP*, the version is *none(3)*.**
ciscoVtpVlanState	.1.3.6.1.4.1.9.9.46.1.3.1.1.2	The state of this *VLAN*. The state *mtuTooBigForDevice* indicates that this device cannot participate in this *VLAN* because the *VLAN's MTU* is larger than the device can support. The state *mtuTooBigForTrunk* indicates that while this *VLAN's MTU* is supported by this device, it is too large for one or more of the device's trunk ports. *operational(1), suspended(2), mtuTooBigForDevice(3), mtuTooBigForTrunk(4)*
ciscoVtpVlanType	.1.3.6.1.4.1.9.9.46.1.3.1.1.3	The type of this *VLAN*. *ethernet(1), fddi(2), tokenRing(3), fddiNet(4), trNet(5), deprecated(6)*
ciscoVtpVlanName	.1.3.6.1.4.1.9.9.46.1.3.1.1.4	The name of this *VLAN*. This name is used as the *ELAN-name* for an *ATM LAN-Emulation* segment of this *VLAN*.

Generic information about the *CDP* process can be found in the *CDP Information* box on the *Node Detail Page* of the device. Information gathered from these OIDs will be stored in the following database table:

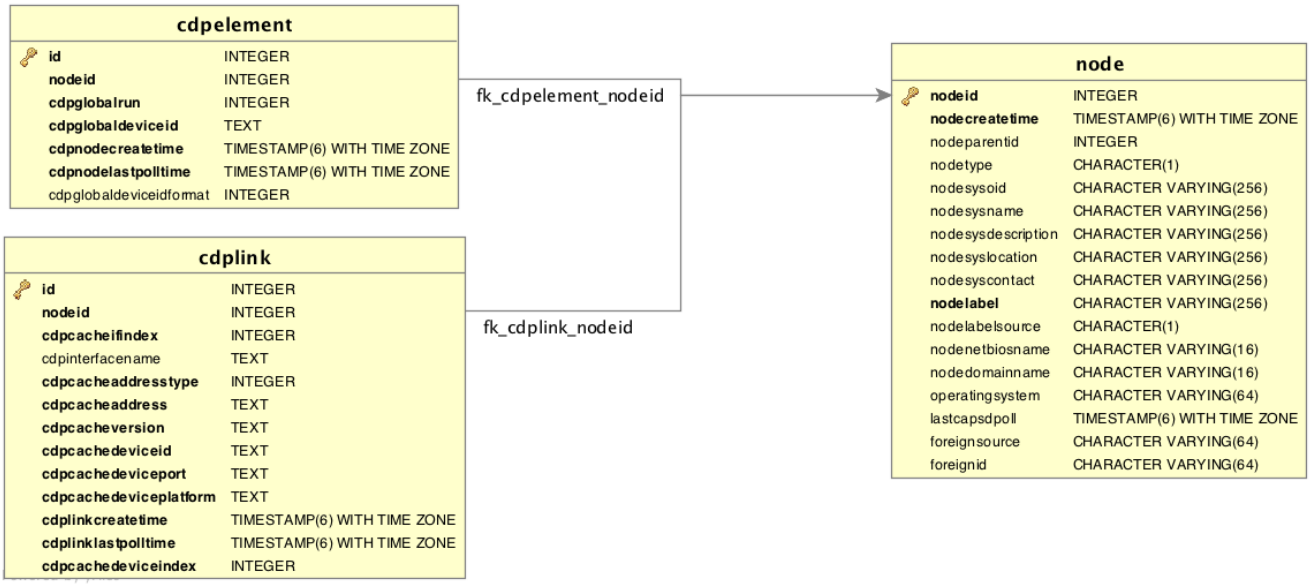


Figure 18. Database tables related to CDP discovery

10.2.3. Transparent Bridge Discovery

Discovering *Layer 2* network links using the *Bridge Forwarding* table requires a special algorithm. To discover *Links* an algorithm based on a scientific paper with the title Topology Discovery for Large Ethernet Networks is implemented. The gathered information is used to classify *Links* in *macLink* and *bridgeLink*. A *macLink* represents a *Link* between a workstation or server identified by a mac address. A *bridgeLink* is a *connection* between backbone ports.

Transparent bridging is not loop free so if you have loops you have to enable the spanning tree protocol that will detect loops and again will put some ports in a *blocking* state to avoid loops. To get links it is necessary to perform some calculations that let us define the *Links*. The following *MIBS* must be supported by the *SNMP agent* to allow *Transparent Bridge Discovery*.

Table 80. Supported MIBS from the Cisco-VTP MIB

Name	OID	Description
vtpVersion	`.1.3.6.1.4.1.9.9.46.1.1.1.0`	The version of VTP in use on the local system. A device will report its version capability and not any particular version in use on the device. If the device does not support *VTP*, the version is *none(3)*.

Table 81. Supported OIDs from the IP-MIB

Name	OID	Description
ipNetToMediaIfIndex	`.1.3.6.1.2.1.4.22.1.1`	The interface on which this entry's equivalence is effective. The layer-2 interface identified by a particular value of this index is the same interface as identified by the same value of *ifIndex*.
ipNetToMediaPhysAddress	`.1.3.6.1.2.1.4.22.1.2`	The media-dependent *physical* address.

ipNetToMediaNetAddress	.1.3.6.1.2.1.4.22.1.3	The *IpAddress* corresponding to the media-dependent *physical* address.
ipNetToMediaType	.1.3.6.1.2.1.4.22.1.4	The type of mapping. Setting this object to the value *invalid(2)* has the effect of invalidating the corresponding entry in the *ipNetToMediaTable*. That is, it effectively dissasociates the interface identified with said entry from the mapping identified with said entry. It is an implementation-specific matter as to whether the agent removes an invalidated entry from the table. Accordingly, management stations must be prepared to receive tabular information from agents that corresponds to entries not currently in use. Proper interpretation of such entries requires examination of the relevant *ipNetToMediaType* object.

Table 82. Supported OIDS from the BRIDGE-MIB

Name	OID	Description
dot1dBaseBridgeAddress	.1.3.6.1.2.1.17.1.1.0	The MAC address used by this bridge when it must be referred to in a unique fashion. It is recommended that this be the numerically smallest MAC address of all ports that belong to this bridge. However it is only required to be unique. When concatenated with *dot1dStpPriority* a unique *BridgeIdentifier* is formed which is used in the *Spanning Tree Protocol*.
dot1dBaseNumPorts	.1.3.6.1.2.1.17.1.2.0	The number of ports controlled by this bridging entity.
dot1dBaseType	.1.3.6.1.2.1.17.1.3.0	Indicates what type of bridging this bridge can perform. If a bridge is actually performing a certain type of bridging this will be indicated by entries in the port table for the given type.
dot1dBasePort	.1.3.6.1.2.1.17.1.4.1.1	The port number of the port for which this entry contains bridge management information.
dot1dPortIfIndex	.1.3.6.1.2.1.17.1.4.1.2	The value of the instance of the *ifIndex* object, defined in *MIB-II*, for the interface corresponding to this port.
dot1dStpProtocolSpecification	.1.3.6.1.2.1.17.2.1.0	An indication of what version of the Spanning Tree Protocol is being run. The value *decLb100(2)* indicates the *DEC LANbridge 100 Spanning Tree protocol*. *IEEE 802.1d* implementations will return *ieee8021d(3)*. If future versions of the *IEEE Spanning Tree Protocol* are released that are incompatible with the current version a new value will be defined.

dot1dStpPriority	.1.3.6.1.2.1.17.2.2	The value of the writeable portion of the *Bridge ID*, i.e., the first two octets of the (8 octet long) *Bridge ID*. The other (last) 6 octets of the *Bridge ID* are given by the value of *dot1dBaseBridgeAddress*.
dot1dStpDesignatedRoot	.1.3.6.1.2.1.17.2.5	The bridge identifier of the root of the spanning tree as determined by the *Spanning Tree Protocol* as executed by this node. This value is used as the *Root Identifier* parameter in all configuration *Bridge PDUs* originated by this node.
dot1dStpRootCost	.1.3.6.1.2.1.17.2.6	The cost of the path to the root as seen from this bridge.
dot1dStpRootPort	.1.3.6.1.2.1.17.2.7	The port number of the port which offers the lowest cost path from this bridge to the root bridge.
dot1dStpPort	.1.3.6.1.2.1.17.2.15.1.1	The port number of the port for which this entry contains Spanning Tree Protocol management information.
dot1dStpPortPriority	.1.3.6.1.2.1.17.2.15.1.2	The value of the priority field which is contained in the first (in network byte order) octet of the (2 octet long) Port ID. The other octet of the Port ID is given by the value of *dot1dStpPort*.
dot1dStpPortState	.1.3.6.1.2.1.17.2.15.1.3	The port's current state as defined by application of the *Spanning Tree Protocol*. This state controls what action a port takes on reception of a frame. If the bridge has detected a port that is malfunctioning it will place that port into the *broken(6)* state. For ports which are disabled (see *dot1dStpPortEnable*), this object will have a value of *disabled(1)*.
dot1dStpPortEnable	.1.3.6.1.2.1.17.2.15.1.4	The enabled/disabled status of the port.
dot1dStpPortPathCost	.1.3.6.1.2.1.17.2.15.1.5	The contribution of this port to the path cost of paths towards the spanning tree root which include this port. 802.1D-1990 recommends that the default value of this parameter be in inverse proportion to the speed of the attached LAN.
dot1dStpPortDesignatedRoot	.1.3.6.1.2.1.17.2.15.1.6	The unique *Bridge Identifier* of the *Bridge* recorded as the *Root* in the *Configuration BPDUs* transmitted by the *Designated Bridge* for the segment to which the port is attached.
dot1dStpPortDesignatedCost	.1.3.6.1.2.1.17.2.15.1.7	The path cost of the *Designated Port* of the segment connected to this port. This value is compared to the *Root Path Cost* field in received bridge *PDUs*.
dot1dStpPortDesignatedBridge	.1.3.6.1.2.1.17.2.15.1.8	The *Bridge Identifier* of the bridge which this port considers to be the *Designated Bridge* for this port's segment.

dot1dStpPortDesignatedPort	.1.3.6.1.2.1.17.2.15.1.9	The *Port Identifier* of the port on the *Designated Bridge* for this port's segment.
dot1dTpFdbAddress	.1.3.6.1.2.1.17.4.3.1.1	A unicast *MAC address* for which the bridge has forwarding and/or filtering information.
dot1dTpFdbPort	.1.3.6.1.2.1.17.4.3.1.2	Either the value '0', or the port number of the port on which a frame having a source address equal to the value of the corresponding instance of *dot1dTpFdbAddress* has been seen. A value of '0' indicates that the port number has not been learned but that the bridge does have some forwarding/filtering information about this address (e.g. in the *dot1dStaticTable*). Implementors are encouraged to assign the port value to this object whenever it is learned even for addresses for which the corresponding value of *dot1dTpFdbStatus* is not *learned(3)*.
dot1dTpFdbStatus	.1.3.6.1.2.1.17.4.3.1.3	The status of this entry. The meanings of the values are: ***other(1)***: none of the following. This would include the case where some other *MIB* object (not the corresponding instance of *dot1dTpFdbPort*, nor an entry in the *dot1dStaticTable*) is being used to determine if and how frames addressed to the value of the corresponding instance of *dot1dTpFdbAddress* are being forwarded. ***invalid(2)***: this entry is not longer valid (e.g., it was learned but has since aged-out), but has not yet been flushed from the table. ***learned(3)***: the value of the corresponding instance of *dot1dTpFdbPort* was learned, and is being used. ***self(4)***: the value of the corresponding instance of *dot1dTpFdbAddress* represents one of the bridge's addresses. The corresponding instance of *dot1dTpFdbPort* indicates which of the bridge's ports has this address. ***mgmt(5)***: the value of the corresponding instance of dot1dTpFdbAddress is also the value of an existing instance of dot1dStaticAddress.

Table 83. Supported OIDS from the Q-BRIDGE-MIB

Name	OID	Description
dot1qTpFdbPort	.1.3.6.1.2.1.17.7.1.2.2.1.2	Either the value *0*, or the port number of the port on which a frame having a source address equal to the value of the corresponding instance of *dot1qTpFdbAddress* has been seen. A value of *0* indicates that the port number has not been learned but that the device does have some forwarding/filtering information about this address (e.g., in the *dot1qStaticUnicastTable*). Implementors are encouraged to assign the port value to this object whenever it is learned, even for addresses for which the corresponding value of *dot1qTpFdbStatus* is not *learned(3)*.

dot1qTpFdbStatus	.1.3.6.1.2.1.17.7.1.2.2.1.3	The status of this entry. The meanings of the values are: ***other(1)***: none of the following. This may include the case where some other MIB object (not the corresponding instance of *dot1qTpFdbPort*, nor an entry in the *dot1qStaticUnicastTable*) is being used to determine if and how frames addressed to the value of the corresponding instance of *dot1qTpFdbAddress* are being forwarded. ***invalid(2)***: this entry is no longer valid (e.g., it was learned but has since aged out), but has not yet been flushed from the table. ***learned(3)***: the value of the corresponding instance of *dot1qTpFdbPort* was learned and is being used. ***self(4)***: the value of the corresponding instance of *dot1qTpFdbAddress* represents one of the device's addresses. The corresponding instance of *dot1qTpFdbPort* indicates which of the device's ports has this address. ***mgmt(5)***: the value of the corresponding instance of *dot1qTpFdbAddress* is also the value of an existing instance of *dot1qStaticAddress*.

Generic information about the *bridge* link discovery process can be found in the *Bridge Information* box on the *Node Detail Page* of the device. Information gathered from this *OID* will be stored in the following database table:

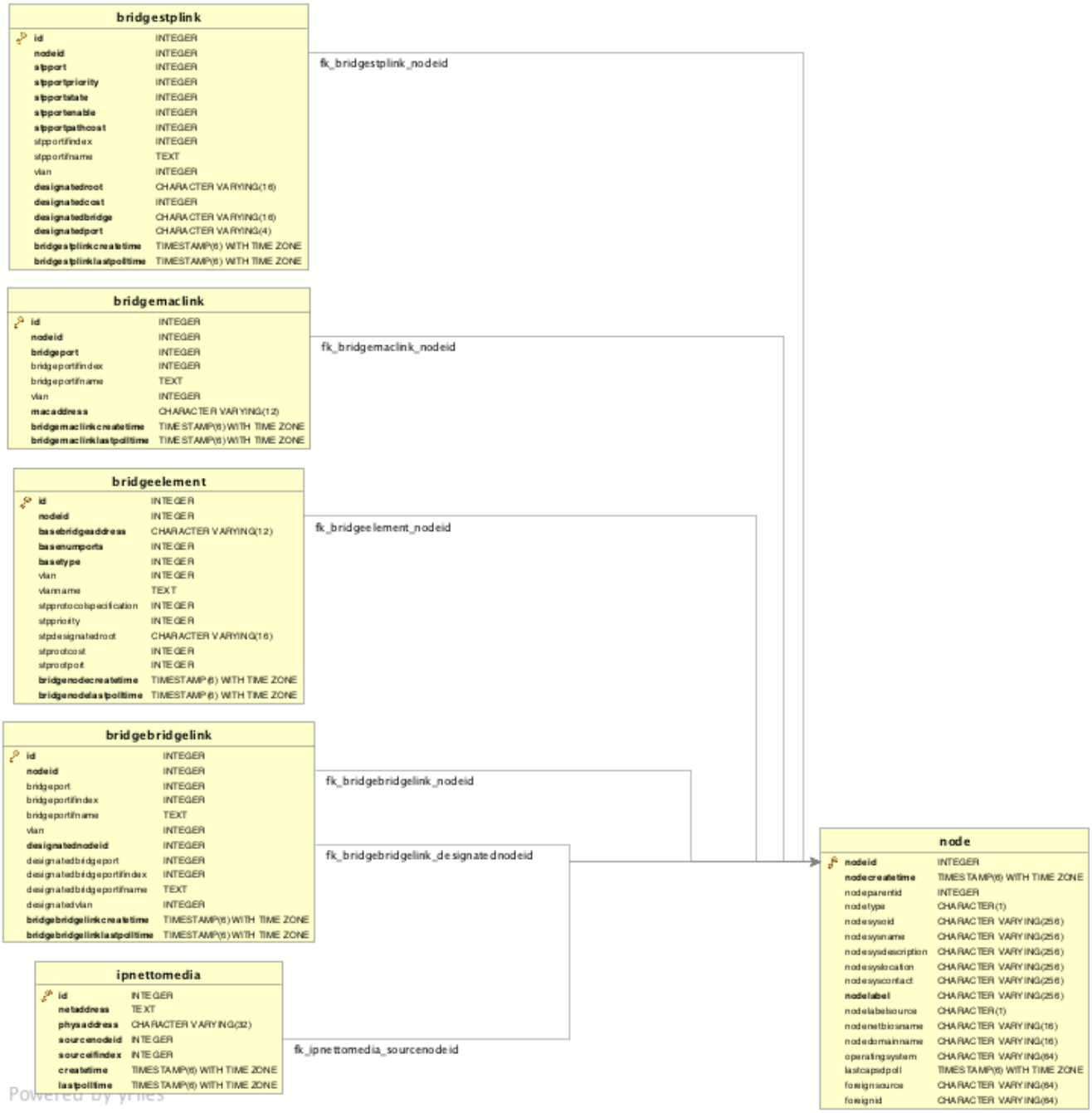


Figure 19. Database tables related to transparent bridge discovery

10.3. Layer 3 Link Discovery

With *Enlinkd* it is possible to get *Links* based on network routing applications. The following routing daemons can be used to provide a discovery of links based *Layer 3* information:

- Open Shortest Path First (OSPF)
- Intermediate System to Intermediate System (IS-IS)

This information is provided by *SNMP Agents* with appropriate *MIB support*. For this reason it is required to have a working *SNMP* configuration running. The link data discovered from *Enlinkd* is provided in the *Topology User Interface* and on the detail page of a node.

10.3.1. OSPF Discovery

The relevant MIBs for OSPF topology are *OSPF-MIB* and *OSPF-TRAP-MIB*. In these MIBs are defined the relevant objects used to find OSPF links, specifically:

- The *Router ID* which, in OSPF, has the same format as an IP address
- But identifies the router independent of its IP address.

Also all the interfaces are identified by their IP addresses. The OSPF links come from the SNMP *ospfNbrTable* defined in OSPF-MIB and this table is in practice persisted in the *ospfLink* table:

Table 84. Supported OIDs from OSPF-MIB

Name	OID	Description
ospfRouterId	.1.3.6.1.2.1.14.1.1.0	A 32-bit integer uniquely identifying the router in the Autonomous System. By convention, to ensure uniqueness, this should default to the value of one of the router's IP interface addresses. This object is persistent and when written the entity **should** save the change to non-volatile storage.
ospfAdminStat	.1.3.6.1.2.1.14.1.2.0	The administrative status of *OSPF* in the router. The value *enabled* denotes that the *OSPF Process* is active on at least one interface; *disabled* disables it on all interfaces. This object is persistent and when written the entity **should** save the change to non-volatile storage.
ospfVersionNumber	.1.3.6.1.2.1.14.1.3.0	The current version number of the *OSPF protocol* is *2*.
ospfAreaBdrRtrStatus	.1.3.6.1.2.1.14.1.4.0	A flag to note whether this router is an *Area Border Router*.
ospfAreaASBdrRtrStatus	.1.3.6.1.2.1.14.1.5.0	A flag to note whether this router is configured as an *Autonomous System Border Router*. This object is persistent and when written the entity **should** save the change to non-volatile storage.
ospfIfIpAddress	.1.3.6.1.2.1.14.7.1.1	The IP address of this *OSPF* interface.
ospfAddressLessIf	.1.3.6.1.2.1.14.7.1.2	For the purpose of easing the instancing of addressed and addressless interfaces; this variable takes the value *0* on interfaces with IP addresses and the corresponding value of *ifIndex* for interfaces having no *IP address*.
ospfNbrIpAddr	.1.3.6.1.2.1.14.10.1.1	The IP address this neighbor is using in its IP source address. Note that, on addressless links, this will not be 0.0.0.0 but the address of another of the neighbor's interfaces.
ospfNbrAddressLessIndex	.1.3.6.1.2.1.14.10.1.2	On an interface having an *IP address*, zero. On addressless interfaces, the corresponding value of *ifIndex* in the *Internet Standard MIB*. On row creation, this can be derived from the instance.

Name	OID	Description
ospfNbrRtrId	.1.3.6.1.2.1.14.10.1.3	A 32-bit integer (represented as a type *IpAddress*) uniquely identifying the neighboring router in the *Autonomous System*.

Table 85. Supported OIDs from IP-MIB

Name	OID	Description
ipAdEntIfIndex	.1.3.6.1.2.1.4.20.1.2	The index value which uniquely identifies the interface to which this entry is applicable. The interface identified by a particular value of this index is the same interface as identified by the same value of the *IF-MIB's ifIndex*.
ipAdEntNetMask	.1.3.6.1.2.1.4.20.1.3	The subnet mask associated with the *IPv4* address of this entry. The value of the mask is an *IPv4* address with all the network bits set to *1* and all the hosts bits set to *0*.

Generic information about the *OSPF* link discovery process can be found in the *OSPF Information* box on the *Node Detail Page* of the device. Information gathered from these OIDs will be stored in the following database table:

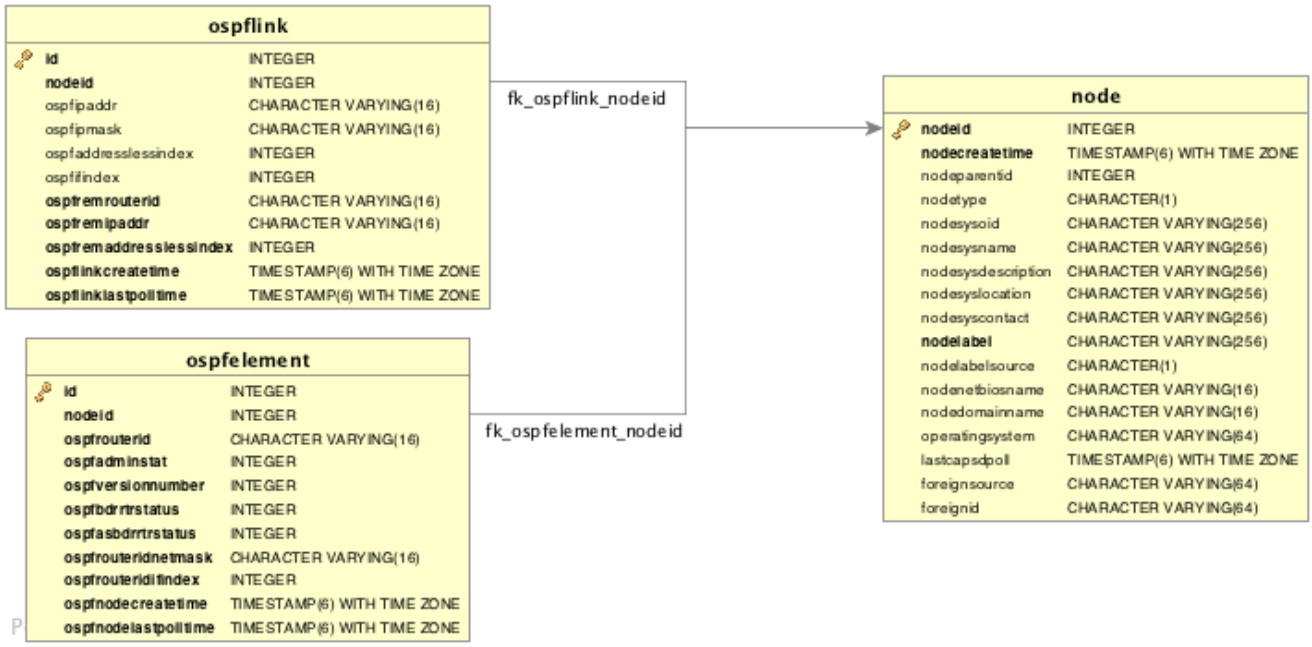


Figure 20. Database tables related to OSPF discovery

10.3.2. IS-IS Discovery

IS-IS Links are found in the *isisISAdjTable* that is defined in *ISIS-MIB* (mib-rfc4444.txt). In this table is found the information needed to find the Adjacency Intermediate System. The information about IS-IS is stored into two tables: *isisElement* and *isisLink*. *isisElement* contains the *ISISSysID*, a unique identifier of the "Intermediate System" (the name for the Router in ISO protocols). Each entry in this SNMP MIB table represents a unidirectional link from the *Intermediate System* that is queried to the *Adjacent Intermediate Systems* running IS-IS and "peering" with the source router. If two routers *IS-A* and *IS-B* support *ISIS-MIB*, then *EnLinkd* will create two link entries in OpenNMS Horizon: one from *IS-A* to *IS-B* (from the *adjtable* of *IS-A*) the complementary link back from *IS-B* to *IS-A* (from the

adjTable of _IS-B). IS-IS links are represented in the *ISIS-MIB* as follows:

Table 86. Supported OIDs from ISIS-MIB

Name	OID	Description
isisSysID	`.1.3.6.1.2.1.138.1.1.1.3.0`	The ID for this Intermediate System. This value is appended to each of the area addresses to form the Network Entity Titles. The derivation of a value for this object is implementation specific. Some implementations may automatically assign values and not permit an SNMP write, while others may require the value to be set manually. Configured values **must** survive an agent reboot.
isisSysAdminState	`.1.3.6.1.2.1.138.1.1.1.8.0`	The administrative state of this Intermediate System. Setting this object to the value `on` when its current value is `off` enables the Intermediate System. Configured values **must** survive an agent reboot.
isisSysObject	`.1.3.6.1.2.1.138.1.1.1`	isisSysObject
isisCircIfIndex	`.1.3.6.1.2.1.138.1.3.2.1.2`	The value of ifIndex for the interface to which this circuit corresponds. This object cannot be modified after creation.
isisCircAdminState	`.1.3.6.1.2.1.138.1.3.2.1.3`	The administrative state of the circuit.
isisISAdjState	`.1.3.6.1.2.1.138.1.6.1.1.2`	The state of the adjacency.
isisISAdjNeighSNPAAddress	`.1.3.6.1.2.1.138.1.6.1.1.4`	The *SNPA address* of the neighboring system.
isisISAdjNeighSysType	`.1.3.6.1.2.1.138.1.6.1.1.5`	The type of the neighboring system.
isisISAdjNeighSysID	`.1.3.6.1.2.1.138.1.6.1.1.6`	The system ID of the neighboring Intermediate System.
isisISAdjNbrExtendedCircID	`.1.3.6.1.2.1.138.1.6.1.1.7`	The 4-byte *Extended Circuit ID* learned from the Neighbor during 3-way handshake, or *0*.

Generic information about the *IS-IS* link discovery process can be found in the *IS-IS Information* box on the *Node Detail Page* of the device. Information gathered from this OIDs will be stored in the following database table:

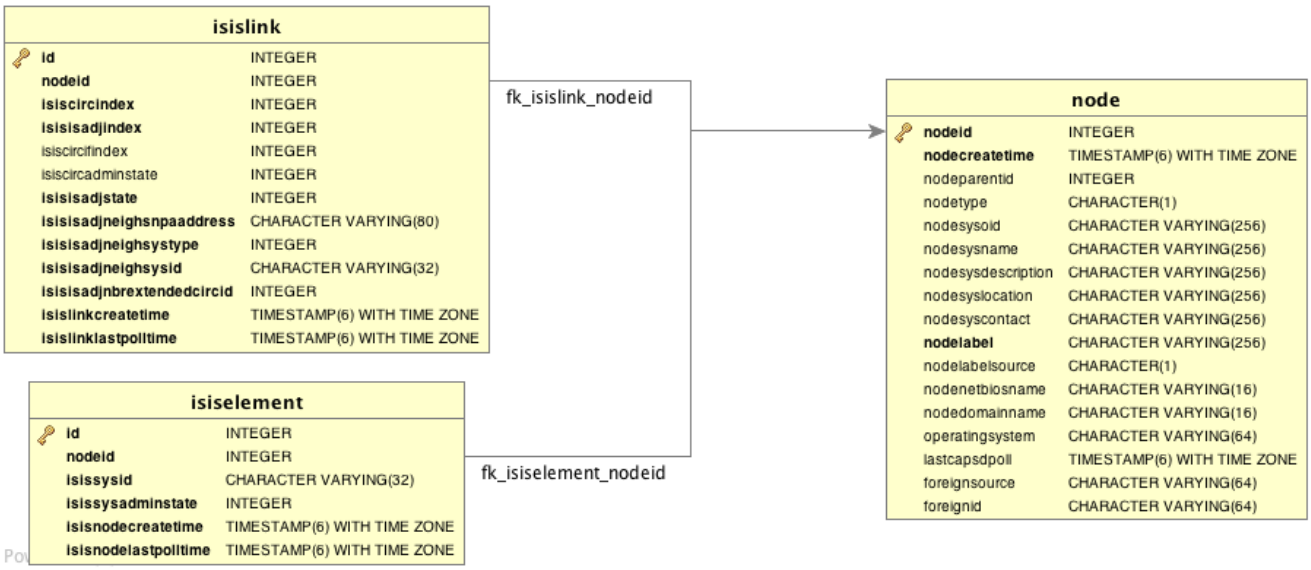


Figure 21. Database tables related to IS-IS discovery

Chapter 11. Operation

11.1. HTTPS / SSL

This chapter covers the possibilities to configure *OpenNMS Horizon* to protect web sessions with HTTPS and also explains how to configure *OpenNMS Horizon* to establish secure connections.

In order to use HTTPS the Java command line tool `keytool` is used. It is automatically shipped with each JRE installation. More details about the `keytool` can be found at the official documentation.

11.1.1. Standalone HTTPS with Jetty

To configure *OpenNMS Horizon* to protect web sessions with HTTPS please refer to the official *OpenNMS Horizon* Wiki article Standalone HTTPS with Jetty.

11.1.2. OpenNMS Horizon as HTTPS client

To establish secure HTTPS connections within Java one has to setup a so called *Java Trust Store*.

The *Java Trust Store* contains all certificates a Java application should trust when making connections as a client to a server.

Setup *Java Trust Store*

To setup the *Java Trust Store* the following command can be issued.

If you do not have a *Java Trust Store* setup yet, it is created automatically.

Import a certificate to the Java Trust Store

```
keytool \
  -import \ ①
  -v \ ②
  -trustcacerts \ ③
  -alias localhost \ ④
  -file localhost.cert \ ⑤
  -keystore /$OPENNMS_HOME/etc/trust-store.jks  ⑥
```

① Define to import a certificate or a certificate chain

② Use verbose output

③ Define to trust certificates from cacerts

④ The alias for the certificate to import, e.g. the common name

⑤ The certificate to import

⑥ The location of the *Java Trust Store*

If you create a new *Java Trust Store* you are asked for a password to protect the *Java Trust Store*. If you update an already existing *Java Trust Store* please enter the password you chose when creating the *Java Trust Store* initially.

Download existing public certificate

To Download an existing public certificate the following command can be issued.

Download an existing public certificate

```
openssl \
  s_client \ ①
  -showcerts \ ②
  -connect localhost:443 \ ③
  -servername localhost \ ④
  < /dev/null \ ⑤
  > localhost.cert ⑥
```

① Use SSL/TLS client functionality of `openssl`.

② Show all certificates in the chain

③ PORT:HOST to connect to, e.g. localhost:443

④ This is optional, but if you are serving multiple certificates under one single ip address you may define a server name, otherwise the `ip of localhost`:PORT certificate is returned which may not match the requested server name (`mail.domain.com`, `opennms.domain.com`, `dns.domain.com`)

⑤ No input

⑥ Where to store the certificate.

Configure OpenNMS Horizon to use the defined *Java Trust Store*

To setup *OpenNMS Horizon* to use the defined *Java Trust Store* the according `javax.net.ssl.trustStore*` properties have to be set. Open `$OPENNMS_HOME/etc/opennms.properties` and add the properties `javax.net.ssl.trustStore` and `javax.net.ssl.trustStorePassword` as shown below.

$OPENNMS_HOME/etc/opennms.properties snippet to define a Java Trust Store

```
javax.net.ssl.trustStore=/$OPENNMS_HOME/etc/trust-store.jks ①
javax.net.ssl.trustStorePassword=change-me ②
```

① The location of the *Java Trust Store*

② The password of the *Java Trust Store*

For more details on the Java build-in SSL System properties have a look at chapter Debugging / Properties.

Each time you modify the *Java Trust Store* you have to restart *OpenNMS Horizon* to have the changes take effect.

11.1.3. Differences between *Java Trust Store* and *Java Key Store*

The *Java Trust Store* is used to determine whether a remote connection should be trusted or not, e.g. whether a remote party is who it claims to be (client use case).

The *Java Key Store* is used to decide which authentication credentials should be sent to the remote host for authentication during SSL handshake (server use case).

For more details, please check the JSSE Reference Guide.

11.1.4. Debugging / Properties

If you encounter issues while using HTTPS it might be useful to enable debugging or use one of the build-in Java System Properties to configure the proper use of SSL.

Table 87. Java build-in System Properties (Source)

System Property Name	Description
`javax.net.ssl.keyStore`	Location of the Java keystore file containing an application process's own certificate and private key. On Windows, the specified pathname must use forward slashes, `/`, in place of backslashes, `\`.
`javax.net.ssl.keyStorePassword`	Password to access the private key from the keystore file specified by `javax.net.ssl.keyStore`. This password is used twice: to unlock the keystore file (store password) and to decrypt the private key stored in the keystore (key password). In other words, the JSSE framework requires these passwords to be identical.
`javax.net.ssl.keyStoreType`	(Optional) For Java keystore file format, this property has the value `jks` (or `JKS`). You do not normally specify this property, because its default value is already `jks`.
`javax.net.ssl.trustStore`	Location of the Java keystore file containing the collection of CA certificates trusted by this application process (trust store). On Windows, the specified pathname must use forward slashes, `/`, in place of backslashes, `\`. If a trust store location is not specified using this property, the Sun JSSE implementation searches for and uses a keystore file in the following locations (in order): `$JAVA_HOME/lib/security/jssecacerts` and `$JAVA_HOME/lib/security/cacerts`
`javax.net.ssl.trustStorePassword`	Password to unlock the keystore file (store password) specified by `javax.net.ssl.trustStore`.

System Property Name	Description
`javax.net.ssl.trustStoreType`	(Optional) For Java keystore file format, this property has the value `jks` (or `JKS`). You do not normally specify this property, because its default value is already `jks`.
`javax.net.debug`	To switch on logging for the SSL/TLS layer, set this property to ssl. More details about possible values can be found here.

11.2. resourcecli: simple resource management tool

Sometimes a user want to list or manually delete collected data (resources) of an *OpenNMS Horizon* instance. When using *RRDTool-* or *JRobin*-based storage this can easily be achieved by traversing the `share/rrd` directory and its subdirectories. The several `.rrd` or `.jrb` files can be listed or deleted for individual nodes. When *Newts*-based storage is used the data is stored and indexed remotely on a *Cassandra* cluster. In this case the cluster must be queried for available resources. For the deletion of resources the data and all generated indexes must be gathered and removed. The *resourcecli* tool simplifies this process and works with *Newts*-based storage as well as with *RRDTool* and *JRobin* files.

11.2.1. Usage

The utility is installed by default and its wrapper script is located in the `${OPENNMS_HOME}/bin` directory.

```
$ cd /path/to/opennms/bin
$ ./resourcecli
```

When invoked without parameters the usage and help information is printed.

The *resourcecli* tool uses sub-commands for the different tasks. Each of these sub-commands provide different options and parameters. The command line tool accepts the following sub-commands.

Sub-command	Description
`list`	Queries an *OpenNMS Horizon* server for available resources.
`show`	Displays details for a given resource.
`delete`	Deletes a given resource and all of its child resources.

The following global options are available in each of the sub-commands of the tool:

Option/Argument	Description	Default
`--help`	Displays help and exit	false
`--username VALUE`	Username for connecting to *OpenNMS Horizon*	admin
`--password VALUE`	Password for connecting to *OpenNMS Horizon*	admin
`--url VALUE`	URL of the *OpenNMS Horizon* instance to connect to	http://localhost:8980/opennms

11.2.2. Sub-command: list

This sub-command is used to query an *OpenNMS Horizon* instance for its available resources. The following example queries the local *OpenNMS Horizon* instance with the credentials `admin/secret`.

```
$ ./resourcecli --username admin --password secret list
node[72]
  node[72].nodeSnmp[]
  node[72].responseTime[192.168.0.2]
node[70]
  node[70].nodeSnmp[]
  node[70].interfaceSnmp[bridge0]
  node[70].interfaceSnmp[bridge1]
  node[70].interfaceSnmp[vlan0-002500fe1bf3]
    node[70].responseTime[50.16.15.18]
  node[70].responseTime[192.168.0.1]

<output omitted>
```

11.2.3. Sub-command: show

This sub-command can be used to show details for a given resource. The following example display details for the resource identified by resourceId `node[70]`.

```
$ ./resourcecli --username admin --password secret show node\[70\]
ID:         node[70]
Name:       70
Label:      MyRouter
Type:       Node
Link:       element/node.jsp?node=70
Parent ID:  null
Children:
  node[70].nodeSnmp[]
  node[70].interfaceSnmp[bridge0]
  node[70].interfaceSnmp[bridge1]
  node[70].interfaceSnmp[vlan0-002500fe1bf3]
    node[70].responseTime[50.16.15.18]
  node[70].responseTime[192.168.0.1]
Attributes:
  External:
  Graphs:
  Strings:
```

The following options are available for the *show* sub-command.

Option/Argument	Description	Default
`<resource>`	The resourceId of the resource to display.	-

11.2.4. Sub-command: delete

This sub-command can be used to delete a given resource and its child resources. The following example deletes the resource identified by resourceId `node[70]`. When successful, this command does not generate any output.

```
$ ./resourcecli --username admin --password secret delete node\[70\]
$
```

The following options are available for the *delete* sub-command.

Option/Argument	Description	Default
`<resource>`	The resourceId of the resource to be deleted.	-

11.3. newts-repository-converter: Rrd/Jrb to Newts migration utility

This utility can be used to migrate existing *RRDTool*- or *JRobin*-based data to a *Newts* cluster. This will be achieved by traversing the `share/rrd` directory and its subdirectories, reading the data and

properties files and persisting this data to *Newts*.

11.3.1. Migration

The following suggestions try to minimize the data collection gap that occur when reconfiguring *OpenNMS Horizon* for a different storage strategy. First, we determine the parameters needed for migration of the existing data. After that, we reconfigure *OpenNMS Horizon* to persists all new collected data to *Newts* storage. Finally, the *Rrd-* or *JRobin*-based data will be converted and persisted to *Newts* using the *newts-repository-converter* utility.

Prerequisites

- Working OpenNMS Horizon installation with *RRDTool-* or *JRobin*-based storage strategy configured.
- Installed and working *Newts* cluster reachable by the *OpenNMS Horizon* instance.

Migration plan

1. Check and write down the values for the following options in your `opennms.properties` file. You will need these information later to invoke the *newts-repository-converter* utility.
   a. File `etc/opennms.properties`:
      - Check for the entry `org.opennms.rrd.storeByGroup` whether `storeByGroup` is enabled.
      - Check for the entry `rrd.base.dir` for the location where *Rrd* or *Jrb* files are stored.
      - Check for the entry `rrd.binary` for the location of the *RRDTool* binary.
   b. File `etc/rrd-configuration.properties`:
      - Check for the entry `org.opennms.rrd.strategyClass` whether `JRobinRrdStrategy` (*JRobin*) or `JniRrdStrategy` / `MultithreadedJniRrdStrategy` (*RRDTool*) is used.
2. Stop your *OpenNMS Horizon* instance.
3. Reconfigure *OpenNMS Horizon* to persist data to *Newts* - so, when correctly configured all new samples will be persisted into *Newts* after *OpenNMS Horizon* is started. Note, that the converter assumes storeByForeignSource to be enabled.
4. Start your *OpenNMS Horizon* instance.
5. Use the *newts-repository-converter* utility to convert the existing data to *Newts* by specifying the options that correspond to the information gathered during step #1.

This procedure will minimize the data collection gap to the time needed to reconfigure *OpenNMS Horizon* for *Newts* storage.

The *newts_converter* utility needs the path to the base directory of your *OpenNMS Horizon* instance for reading the configuration files. For instance the utility needs the datasource configuration during the migration process to query the database to lookup node data.

11.3.2. Usage

The utility is installed by default and its wrapper script is located in the `${OPENNMS_HOME}/bin` directory.

```
$ cd /path/to/opennms/bin
$ ./newts-repository-converter
```

When invoked without parameters the usage and help information is printed.

The *newts-repository-converter* tool provide the following options and parameters:

Short-option	Long-option	Description	Default
`h`	`help`	Prints help and usage information	false
`o`	`onms-home`	*OpenNMS Horizon* Home Directory	/opt/opennms
`r`	`rrd-dir`	The path to the RRD data	ONMS-HOME/share/rrd
`t`	`rrd-tool`	Whether to use rrdtool or JRobin	
`T`	`rrd-binary`	The binary path to the rrdtool command (only used if rrd-tool is set)	/usr/bin/rrdtool
`s`	`store-by-group`	Whether store by group was enabled or not	
`n`	`threads`	Number of conversion threads	defaults to number of CPUs

11.3.3. Example 1: convert Rrd-based data with storeByGroup enabled

The following example shows how to convert *RRDTool*-based data that was stored with `storeByGroup` enabled. The OpenNMS Horizon home is `/opt/opennms`, the data directory is `/opt/opennms/share/rrd` and the *RRDTool* binary located at `/usr/local/bin/rrdtool`. This program call will use 16 concurrent threads to convert the *Rrd* files.

```
$ ./newts-repository-converter -t true -s true -T /usr/local/bin/rrdtool -n 16
<output omitted>
```

11.3.4. Example 2: convert JRobin-based data with storeByGroup disabled

The following example shows how to convert *JRobin*-based data located in the directory `/mnt/opennms/rrd` that was collected with `storeByGroup` disabled. This program call will use 8 concurrent threads to convert the *Jrb* files.

```
$ ./newts-repository-converter -t false -s false -r /mnt/opennms/rrd -n 8
<output omitted>
```

11.4. Newts

This section describes how to configure *OpenNMS Horizon* to use *Newts* and how to use *OpenNMS Horizon* to monitor your Cassandra cluster.

11.4.1. Configuration

Enabling Newts

OpenNMS Horizon can be configured to use Newts by setting the following property in in `${OPENNMS_HOME}/etc/opennms.properties`:

```
org.opennms.timeseries.strategy=newts
```

It is also highly recommended that resources stored in *Newts* are referenced by their foreign source and foreign ID, as opposed to their database ID. To this end, the following property should also be set in the same file:

```
org.opennms.rrd.storeByForeignSource=true
```

With these set, *OpenNMS Horizon* will begin persisting metrics using the *Newts* engine when restarted.

Additional configuration options are presented in the next section.

Configuration Reference

The following properties, found in `${OPENNMS_HOME}/etc/opennms.properties`, can be used to configure and tune *Newts*.

General

Name	Default	Description
`org.opennms.newts.config.keyspace`	`newts`	Name of the keyspace to use.
`org.opennms.newts.config.hostname`	`localhost`	IP address or hostnames of the Cassandra nodes. Multiple hosts can be separated by a comma.
`org.opennms.newts.config.port`	`9042`	CQL port used to connect to the Cassandra nodes.
`org.opennms.newts.config.username`	`cassandra`	Username to use when connceting to Cassandra via CQL.

Name	Default	Description
org.opennms.newts.config.password	cassandra	Password to use when connceting to Cassandra via CQL.
org.opennms.newts.config.read_consistency	ONE	Consistency level used for *read* operations. See Configuring data consistency for a list of available options.
org.opennms.newts.config.write_consistency	ANY	Consistency level used for *write* operations. See Configuring data consistency for a list of available options.
org.opennms.newts.config.max_batch_size	16	Maximum number of records to insert in a single transaction. Limited by the size of the Cassandra cluster's batch_size_fail_threshold_in_kb property.
org.opennms.newts.config.ring_buffer_size	8192	Maximum number of records that can be held in the ring buffer. Must be a power of two.
org.opennms.newts.config.writer_threads	16	Number of threads used to pull samples from the ring buffer and insert them into Newts.
org.opennms.newts.config.ttl	31540000	Number of seconds after which samples will automatically be deleted. Defaults to one year.
org.opennms.newts.config.resource_shard	604800	Duration in seconds for which samples will be stored at the same key. Defaults to 7 days in seconds.
org.opennms.newts.query.minimum_step	30000	Minimum step size in milliseconds. Used to prevent large queries.
org.opennms.newts.query.interval_divider	2	If no interval is specified in the query, the step will be divided into this many intervals when aggregating values.
org.opennms.newts.query.heartbeat	450000	Duration in milliseconds. Used when no heartbeat is specified. Should generally be 1.5x your largest collection interval.
org.opennms.newts.query.parallelism	Number of cores	Maximum number of threads that can be used to compute aggregates. Defaults to the number of available cores.
org.opennms.newts.config.cache.strategy	See bellow	Canonical name of the class used for resource level caching. See the table bellow for all of the available options.
org.opennms.newts.config.cache.max_entries	8192	Maximum number of records to keep in the cache when using an in-memory caching strategy.

Available caching strategies include:

Name	Class	Default
In-Memory Cache	org.opennms.netmgt.newts.support.GuavaSearchableResourceMetadataCache	Y

Name	Class	Default
Redis-based Cache	`org.opennms.netmgt.newts.support.RedisResourceMetadataCache`	N

Redis Cache

When enabled, the following options can be used to configure the Redis-based cache.

Name	Default	Description
`org.opennms.newts.config.cache.redis_hostname`	`localhost`	IP address of hostname of the *Redis* server.
`org.opennms.newts.config.cache.redis_port`	`6379`	TCP port used to connect to the *Redis* server.

Recommendations

You will likely want to change the values of `cache.max_entries` and the `ring_buffer_size` to suit your installation.

Meta-data related to resources are cached in order to avoid writing redundant records in *Cassandra*. If you are collecting data from a large number of resources, you should increase the `cache.max_entries` to reflect the number of resources you are collecting from, with a suitable buffer.

The samples gathered by the collectors are temporarily stored in a ring buffer before they are persisted to *Cassandra* using *Newts*. The value of the `ring_buffer_size` should be increased if you expect large peaks of collectors returning at once or latency in persisting these to *Cassandra*. However, note that the memory used by the ring buffer is reserved, and larger values may require an increased heap size.

11.4.2. Cassandra Monitoring

This section describes some of the metrics *OpenNMS Horizon* collects for monitoring an *Cassandra* database infrastructure. To enable the data collection, the node has to be in the *Surveillance Category* named *Cassandra21*.

The data collection is bound to the agent IP interface with the service name *JMX-Cassandra*. The *JMXCollector* is used to retrieve the *MBean* entities from the *Cassandra* node.

Client Connections

The number of active client connections from `org.apache.cassandra.metrics.Client` are collected:

Name	Description
`connectedNativeClients`	Metrics for connected native clients
`connectedThriftClients`	Metrics for connected thrift clients

Compaction Bytes

The following compaction manager metrics from `org.apache.cassandra.metrics.Compaction` are collected:

Name	Description
`BytesCompacted`	Number of bytes compacted since node started

Compaction Tasks

The following compaction manager metrics from `org.apache.cassandra.metrics.Compaction` are collected:

Name	Description
`CompletedTasks`	Estimated number of completed compaction tasks
`PendingTasks`	Estimated number of pending compaction tasks

Storage Load

The following storage load metrics from `org.apache.cassandra.metrics.Storage` are collected:

Name	Description
`Load`	Total disk space (in bytes) used by this node

Storage Exceptions

The following storage exception metrics from `org.apache.cassandra.metrics.Storage` are collected:

Name	Description
`Exceptions`	Number of unhandled exceptions since start of this *Cassandra* instance

Dropped Messages

Measurement of messages that were *DROPPABLE*. These ran after a given timeout set per message type so was thrown away. In *JMX* these are accessible via `org.apache.cassandra.metrics.DroppedMessage`. The number of dropped messages in the different message queues are good indicators whether a cluster can handle its load.

Name	Stage	Description
`Mutation`	*MutationStage*	If a write message is processed after its timeout (write_request_timeout_in_ms) it either sent a failure to the client or it met its requested consistency level and will relay on hinted handoff and read repairs to do the mutation if it succeeded.

Name	Stage	Description
`Counter_Mutation`	*MutationStage*	If a write message is processed after its timeout (write_request_timeout_in_ms) it either sent a failure to the client or it met its requested consistency level and will relay on hinted handoff and read repairs to do the mutation if it succeeded.
`Read_Repair`	*MutationStage*	Times out after write_request_timeout_in_ms.
`Read`	*ReadStage*	Times out after read_request_timeout_in_ms. No point in servicing reads after that point since it would of returned error to client.
`Range_Slice`	*ReadStage*	Times out after range_request_timeout_in_ms.
`Request_Response`	*RequestResponseStage*	Times out after request_timeout_in_ms. Response was completed and sent back but not before the timeout

Thread pools

Apache Cassandra is based on a so called *Staged Event Driven Architecture* (SEDA). This seperates different operations in stages and these stages are loosely coupled using a messaging service. Each of these components use queues and thread pools to group and execute their tasks. The documentation for *Cassandra* Thread pool monitoring is originated from Pythian Guide to Cassandra Thread Pools.

Table 88. Collected metrics for Thread Pools

Name	Description
`ActiveTasks`	Tasks that are currently running
`CompletedTasks`	Tasks that have been completed
`CurrentlyBlockedTasks`	Tasks that have been blocked due to a full queue
`PendingTasks`	Tasks queued for execution

Memtable FlushWriter

Sort and write *memtables* to disk from `org.apache.cassandra.metrics.ThreadPools`. A vast majority of time this backing up is from over running disk capability. The sorting can cause issues as well however. In the case of sorting being a problem, it is usually accompanied with high load but a small amount of actual flushes (seen in cfstats). Can be from huge rows with large column names, i.e. something inserting many large values into a *CQL* collection. If overrunning disk capabilities, it is recommended to add nodes or tune the configuration.

Alerts: pending > 15 | | blocked > 0

Memtable Post Flusher

Operations after flushing the *memtable*. Discard commit log files that have had all data in them in *sstables*. Flushing non-cf backed secondary indexes.

Alerts: pending > 15 || blocked > 0

Anti Entropy Stage

Repairing consistency. Handle repair messages like merkle tree transfer (from Validation compaction) and streaming.

Alerts: pending > 15 || blocked > 0

Gossip Stage

Post 2.0.3 there should no longer be issue with pending tasks. Instead monitor logs for a message:

```
Gossip stage has {} pending tasks; skipping status check ...
```

Before that change, in particular older versions of 1.2, with a lot of nodes (100+) while using vnodes can cause a lot of CPU intensive work that caused the stage to get behind. Been known to of been caused with out of sync schemas. Check *NTP* working correctly and attempt `nodetool resetlocalschema` or the more drastic deleting of system column family folder.

Alerts: pending > 15 || blocked > 0

Migration Stage

Making schema changes

Alerts: pending > 15 || blocked > 0

MiscStage

Snapshotting, replicating data after node remove completed.

Alerts: pending > 15 || blocked > 0

Mutation Stage

Performing a local including:

- insert/updates
- Schema merges
- commit log replays
- hints in progress

Similar to ReadStage, an increase in pending tasks here can be caused by disk issues, over loading a system, or poor tuning. If messages are backed up in this stage, you can add nodes, tune hardware and configuration, or update the data model and use case.

Alerts: pending > 15 || blocked > 0

Read Stage

Performing a local read. Also includes deserializing data from row cache. If there are pending values this can cause increased read latency. This can spike due to disk problems, poor tuning, or over loading your cluster. In many cases (not disk failure) this is resolved by adding nodes or tuning the system.

Alerts: pending > 15 || blocked > 0

Request Response Stage

When a response to a request is received this is the stage used to execute any callbacks that were created with the original request.

Alerts: pending > 15 || blocked > 0

Read Repair Stage

Performing read repairs. Chance of them occurring is configurable per column family with `read_repair_chance`. More likely to back up if using `CL.ONE` (and to lesser possibly other `non-CL.ALL` queries) for reads and using multiple data centers. It will then be kicked off asynchronously outside of the queries feedback loop. Note that this is not very likely to be a problem since does not happen on all queries and is fast providing good connectivity between replicas. The repair being droppable also means that after `write_request_timeout_in_ms` it will be thrown away which further mitigates this. If pending grows attempt to lower the rate for high read `CFs`.

Alerts: pending > 15 || blocked > 0

JVM Metrics

Some key metrics from the running Java virtual machine are also collected:

java.lang:type=Memory
: The memory system of the Java virtual machine. This includes heap and non-heap memory

java.lang:type=GarbageCollector,name=ConcurrentMarkSweep
: Metrics for the garbage collection process of the Java virtual machine

If you use *Apache Cassandra* for running *Newts* you can also enable additional metrics for the *Newts* keyspace.

11.4.3. Newts Monitoring

This section describes the metrics *OpenNMS Horizon* collects for monitoring the *Newts* keyspace from `org.apache.cassandra.metrics.Keyspace` on an *Cassandra* node. To enable the data collection, the node has to be in the *Surveillance Categories* named *Cassandra21* and *Newts*.

The data collection is bound to the agent IP interface with the service name *JMX-Cassandra-Newts*. The *JMXCollector* is used to retrieve the *MBean* entities from the *Cassandra* node.

All Memory Table Data Size

Name	Description
`AllMemtablesLiveDataSize`	Total amount of live data stored in the memtables (2i and pending flush memtables included) that resides off-heap, excluding any data structure overhead
`AllMemtablesOffHeapDataSize`	Total amount of data stored in the memtables (2i and pending flush memtables included) that resides off-heap.
`AllMemtablesOnHeapDataSize`	Total amount of data stored in the memtables (2i and pending flush memtables included) that resides on-heap.

Memtable Switch Count

Name	Description
`MemtableSwitchCount`	Number of times flush has resulted in the memtable being switched out.

Memtable Columns Count

Name	Description
`MemtableColumnsCount`	Total number of columns present in the memtable.

Memory Table Data Size

Name	Description
`MemtableLiveDataSize`	Total amount of live data stored in the memtable, excluding any data structure overhead
`MemtableOffHeapDataSize`	Total amount of data stored in the memtable that resides off-heap, including column related overhead and partitions overwritten.
`MemtableOnHeapDataSize`	Total amount of data stored in the memtable that resides on-heap, including column related overhead and partitions overwritten.

Read and Write Latency

Name	Description
`ReadTotalLatency`	Local read metrics.
`WriteTotalLatency`	Local write metrics.

Range Latency

Name	Description
RangeLatency 99th Percentile	Local range slice metrics 99th percentile.

Latency

Name	Description
CasCommitTotalLatency	
CasPrepareTotalLatency	
CasProposeTotalLatency	

Bloom Filter Disk Space

Name	Description
BloomFilterDiskSpaceUsed	Disk space used by bloom filter

Bloom Filter Off Heap Memory

Name	Description
BloomFilterOffHeapMemoryUsed	Off heap memory used by bloom filter

Newts Memory Used

Name	Description
CompressionMetadataOffHeapMemoryUsed	Off heap memory used by compression meta data
IndexSummaryOffHeapMemoryUsed	Off heap memory used by index summary

Pending

Name	Description
PendingCompactions	Estimate of number of pending compactions for this column family
PendingFlushes	Estimated number of tasks pending for this column family

Disk Space

Name	Description
TotalDiskSpaceUsed	Total disk space used by *SSTables* belonging to this column family including obsolete ones waiting to be garbage collected.
LiveDiskSpaceUsed	Disk space used by *SSTables* belonging to this column family

Chapter 12. System Properties

The global behavior of *OpenNMS* is configured with Property files. Configuration can have also effect on the *Java Virtual Machine* underneath *OpenNMS*. Changes in these property files require a restart of *OpenNMS*. The configuration files can be found in `${OPENNMS_HOME}/etc`.

The priority for *Java system properties* is as follows:

1. Those set via the *Java* command line i.e. in `opennms.conf` via `ADDITIONAL_MANAGER_OPTIONS`
2. `opennms.properties.d/*.properties`
3. `opennms.properties`
4. `libraries.properties`
5. `rrd-configuration.properties`
6. `bootstrap.properties`

Property files in `opennms.properties.d/` are sorted alphabetically.

To avoid conflicts with customized configurations, all custom properties can be added to one or more files in `${OPENNMS_HOME}/etc/opennms.properties.d/`. It is recommended to avoid modification of OpenNMS properties from the default installation. Create dedicated files with your customized properties in `opennms.properties.d/`.

Chapter 13. Ticketing

The ticketing integration allows *OpenNMS Horizon* to create trouble tickets in external systems. Tickets can be created and updated in response to new and/or resolved alarms.

13.1. JIRA Ticketing Plugin

The *JIRA Ticketing Plugin* is used to create JIRA Issues in response to *OpenNMS Horizon* alarms.

13.1.1. Setup

First, you'll need to install the `opennms-plugin-ticketer-jira` package for your system. The JIRA ticketing plugin and it's dependencies are not part of the core packages.

Now, in order to enable the plugin start by setting following property in `${OPENNMS_HOME}/etc/opennms.properties`:

```
opennms.ticketer.plugin=org.opennms.netmgt.ticketd.OSGiBasedTicketerPlugin
```

Configure the plugin options by setting the following properties in `${OPENNMS_HOME}/etc/jira.properties`:

Name	Description
`jira.host`	JIRA Server Url
`jira.username`	Username
`jira.password`	Password
`jira.project`	Project Code
`jira.type`	Issue Type ID to use when opening new issues
`jira.resolve`	Name of the transition to use when resolving issues
`jira.reopen`	Name of the transition to use when re-opening issues
`jira.status.open`	Comma-separated list of JIRA status names for which the ticket should be considered 'Open'
`jira.status.closed`	Comma-separated list of JIRA status names for which the ticket should be considered 'Closed'
`jira.status.cancelled`	Comma-separated list of JIRA status names for which the ticket should be considered 'Cancelled'

The transition names for `resolve` and `reopen` are typically found on buttons when looking at the ticket in JIRA

See https://confluence.atlassian.com/display/JIRA050/Finding+the+Id+for+Issue+Types for determining the appropriate issue type id.

Next, add `jira-troubleticketer` to the `featuresBoot` property in the `${OPENNMS_HOME}/etc/org.apache.karaf.features.cfg`

Restart *OpenNMS Horizon*.

When *OpenNMS Horizon* has started again, login to the *Karaf Shell* and install the feature:

```
features:install jira-troubleticketer
```

The plugin should be ready to use. When troubleshooting, consult the following log files:

- ${OPENNMS_HOME}/data/log/karaf.log
- ${OPENNMS_HOME}/logs/trouble-ticketer.log

13.2. TSRM Ticketing Plugin

The *TSRM Ticketing Plugin* is used to create TSRM incidents in response to *OpenNMS Horizon* alarms.

13.2.1. Setup

In order to enable the plugin start by setting following property in `${OPENNMS_HOME}/etc/opennms.properties`:

```
opennms.ticketer.plugin=org.opennms.netmgt.ticketd.OSGiBasedTicketerPlugin
```

Configure the plugin options by setting the following properties in `${OPENNMS_HOME}/etc/tsrm.properties`:

Name	Description
`tsrm.url`	TSRM Endpoint URL
`tsrm.ssl.strict`	Strict SSL Check (true/false)
`tsrm.status.open`	TSRM status for open ticket
`tsrm.status.close`	TSRM status for close ticket

Next, add `tsrm-troubleticketer` to the `featuresBoot` property in the `${OPENNMS_HOME}/etc/org.apache.karaf.features.cfg`

Restart *OpenNMS*.

When *OpenNMS* has started again, login to the *Karaf Shell* and install the feature:

```
features:install tsrm-troubleticketer
```

The plugin should be ready to use. When troubleshooting, consult the following log files:

- ${OPENNMS_HOME}/data/log/karaf.log
- ${OPENNMS_HOME}/logs/trouble-ticketer.log

13.2.2. Mapping OpenNMS Ticket with TSRM Incident

Following tables shows mapping between OpenNMS ticket and TSRM Incident

Ticket Field	TSRM Incident Field
id	TICKETID
state	STATUS
summary	DESCRIPTION
details	DESCRIPTIONLONGDESCRIPTION
user	REPORTEDBY

Below fields are not part of Ticket, they have to be added as attributes.

Ticket Field	TSRM Incident Field
affectedPerson	AFFECTEDPERSON
assetNum	ASSETNUM
classId	CLASS
classStructureId	CLASSSTRUCTUREID
commodity	COMMODITY
location	LOCATION
ownerGroup	OWNERGROUP
shsCallerType	SHSCALLERTYPE
shsReasonForOutage	SHSREASONFOROUTAGE
shsResolution	SHSRESOLUTION
shsRoomNumber	SHSROOMNUMBER
siteId	SITEID
source	source
statusIface	STATUSIFACE

Chapter 14. Plugin Manager

With the introduction of *Karaf* as an *OSGi* application container, *OpenNMS Horizon* now has the ability to install or upgrade features on top of a running instance of *OpenNMS Horizon*. In addition, the new distributed *OSGi* architecture allows an *OpenNMS Horizon* system to be deployed as multiple software modules each running in their own *Karaf* instance.

The *OpenNMS Horizon Plugin Manager* provides a unified interface for managing the lifecycle of optional *OSGi* plugins installed in *OpenNMS Horizon* or in any *Karaf* instances which it manages. This need not be limited to *Karaf* instances running *OpenNMS Horizon* but can also be used to deploy modules to *Karaf* instances running user applications.

In addition to managing the installation of *OSGi* features, the *Plugin Manager* also allows the installation of licence keys which can be used to enable features for a particular instance of *OpenNMS Horizon*. Although the *OpenNMS Horizon* platform remains open source, this provides a mechanism for third parties developing features on top of the *OpenNMS Horizon* platform to manage access to their software.

The *Plugin Manager* also provides a mechanism for a separate 'app-store' or Available Plugins Server to be used to deliver these new features and / or licences into a particular *OpenNMS Horizon* instance. It is also possible to deliver software without access to the internet using the traditional *Karaf* Kar/RPM deployment model. In this case a number of features can be delivered together in a single software package but each only enabled at run time using the Plugin Manager.

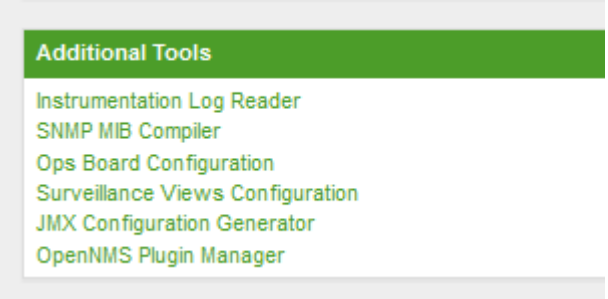
OpenNMS Horizon plugins are standard *Karaf* features with additional metadata which describes the feature and the licence (if any) required. A plugin requiring a licence will not start if a valid licence string is not also installed.

Note that *Karaf*'s features mechanism has not been modified in any way. The Plugin Manager simply provides a user front end and additional metadata for features. Features can be installed from remote maven repositories or *Kar* files placed in the deploy directory depending on how the *Karaf* configuration is set up. The standard *OpenNMS Horizon* configuration has no remote maven access enabled for *Karaf* and features must be locally provisioned as a *Kar* or an *RPM* before being enabled with the *Plugin Manager*.

This guide describes how to deploy and manage plugins using the *Plugin Manager*. A separate plugin developer's guide is provided for those wishing to write their own plugins.

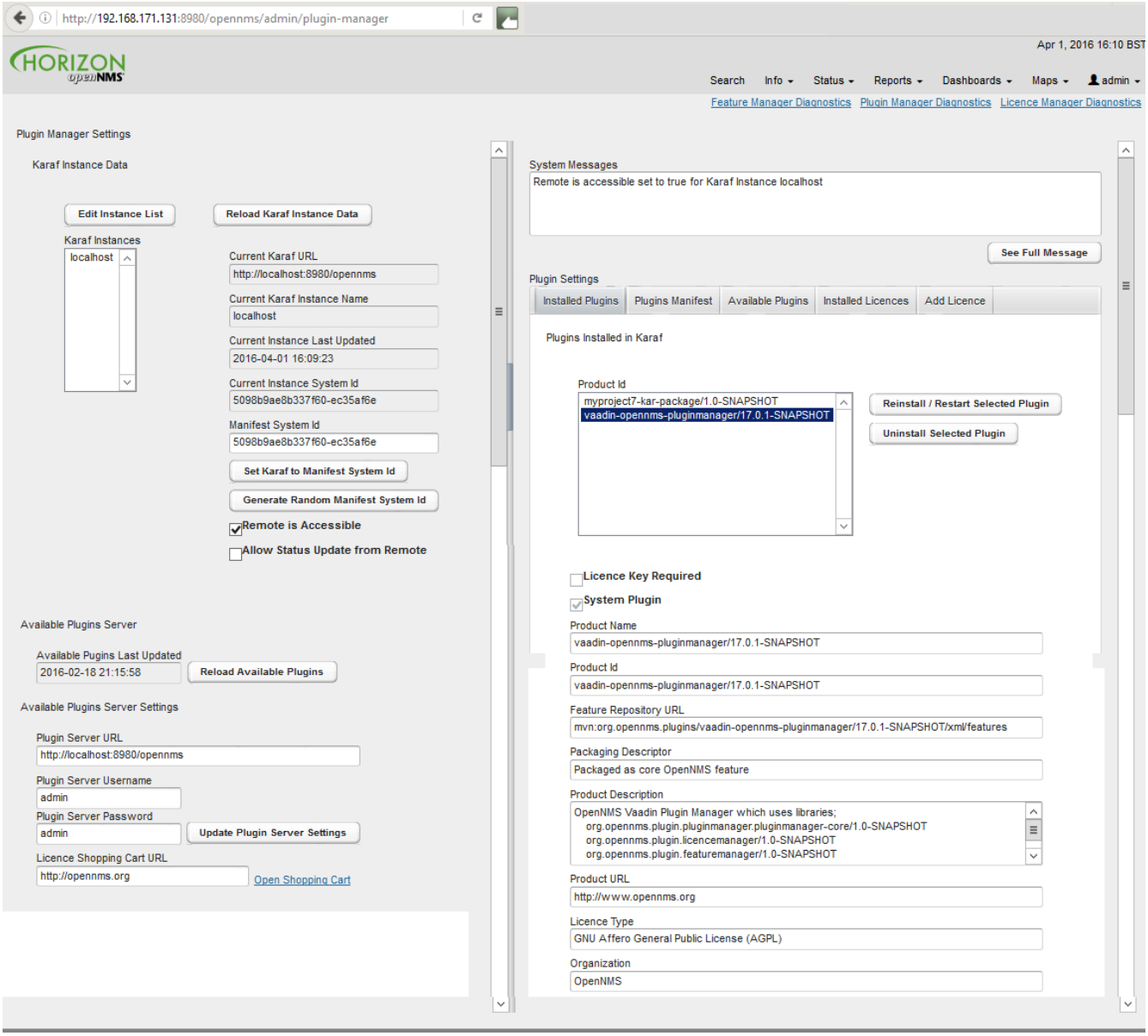
14.1. Plugin Manager UI panel

The *Plugin Manager* is accessed as an entry in the *Additional Tools* panel of the *OpenNMS Horizon Admin Gui*.



The *Plugin Manager* administration page is split into six main areas as illustrated below.

1. Top Left is the *Karaf* Instance data panel which lists the *Karaf* instances known to the *Plugin Manager*. When a *Karaf* instance is selected, the data on the rest of the page refers to the selected instance.
2. Bottom Left is the *Available Plugins Server Panel* which is used to set the address and passwords to access the *Available Plugins Server* and / or the list of locally available plugins provided by a *Kar* or *RPM*.
3. Top Right, just below the main *OpenNMS Horizon* menu bar are links to three diagnostic pages which can help test the *ReST* interface to remote *Karaf* Instances.
4. Middle Right is a messages panel which reports the status of any operations. If an operation fails, the full error message can be viewed by pressing the error message button.
5. Bottom Right is a tabbed panel which reflects the status of the plugins and licences installed in the *Karaf* instance selected by the *Karaf* Instance data panel.



14.2. Setting Karaf Instance Data

The *Karaf* instances known to the *Plugin Manager* are listed in the *Karaf* Instance data panel. 'Localhost' refers to the local *OpenNMS Horizon* server and is always an option in the panel. The *Karaf* instance data is persisted locally and should be refreshed from remote sources using the reload *Karaf* instance data button before changes are made.

Each *Karaf* instance must have a unique system id which is used to update its configuration and also to validate its licences. The system id it must be unique and included a checksum. A new random system id can be generated for a *Karaf* instance using a button on the panel.

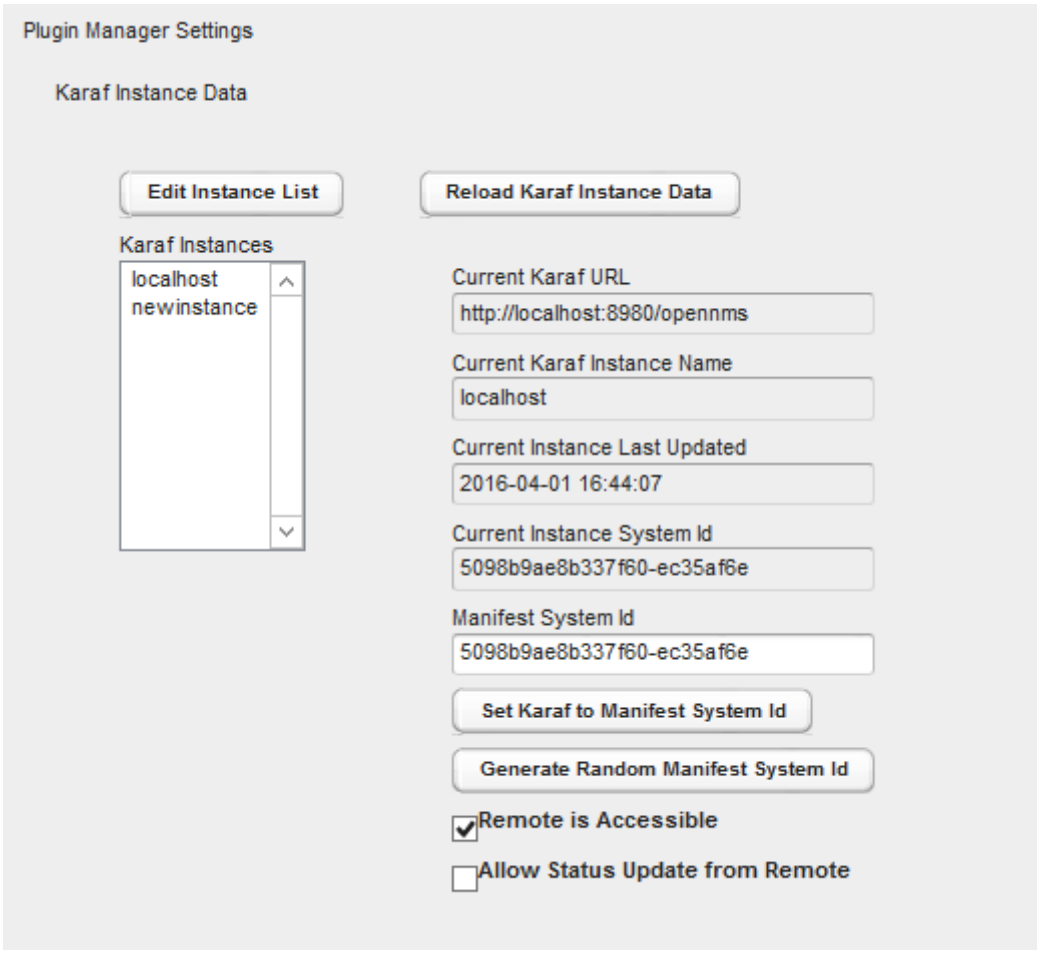
In most situations the remote *Karaf* instance can be accessed from the *OpenNMS Horizon Plugin Manager*. However in many cases, the remote *Karaf* will be behind a firewall in which case it must initiate the communications to request its configuration and supply an update on its status.

The 'Remote is Accessible' field tells the *Plugin Manager* which mode of operation is in use.

Remote request of configuration is not yet fully implemented and will be completed in a future release.

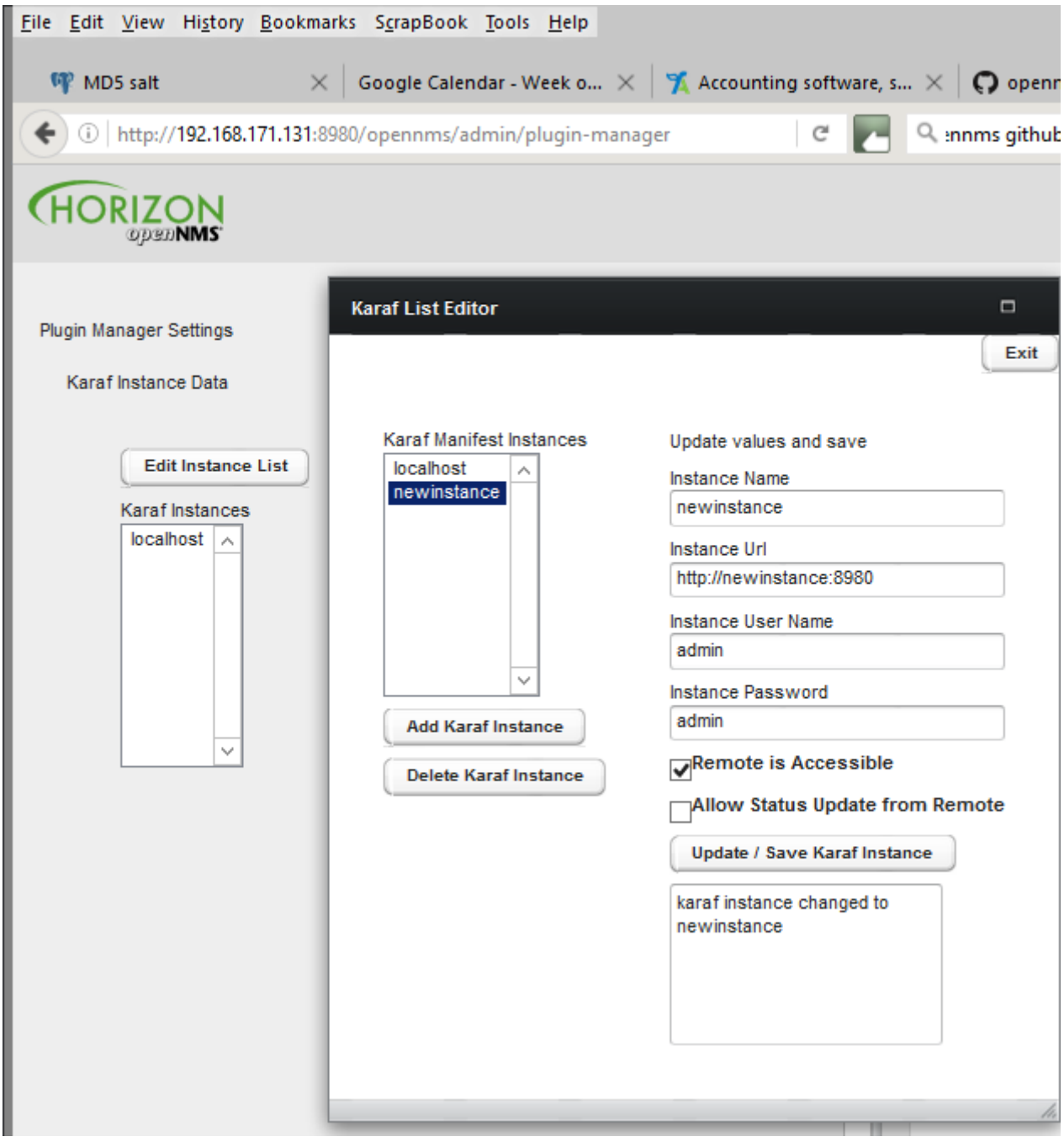
Table 89. Karaf Instance Fields

Field Name	Description
Instance Name	host Name of the *Karaf* instance
Karaf URL	URL used to access the *Karaf Plugin Manager* ReST API
Current Instance System ID	The system ID currently installed in the *Karaf* system
Manifest System ID	The system ID to be provisioned in the *Karaf* system
Remote is Accessible	If ticked 'true', the *Plugin Manager* will try and contact the remote *Karaf* instance using the URL. If not ticked (i.e. false), the remote *Karaf* instance must request its configuration.
Allow Status Update from Remote	Allow the remote *Karaf* instance to request an update to its remote configuration from the locally held manifest and at the same time to update its status.



14.3. Manually adding a managed *Karaf* instance

The list of *Karaf* instances can be modified using the *Karaf* instance editor illustrated below. The same fields apply as above.

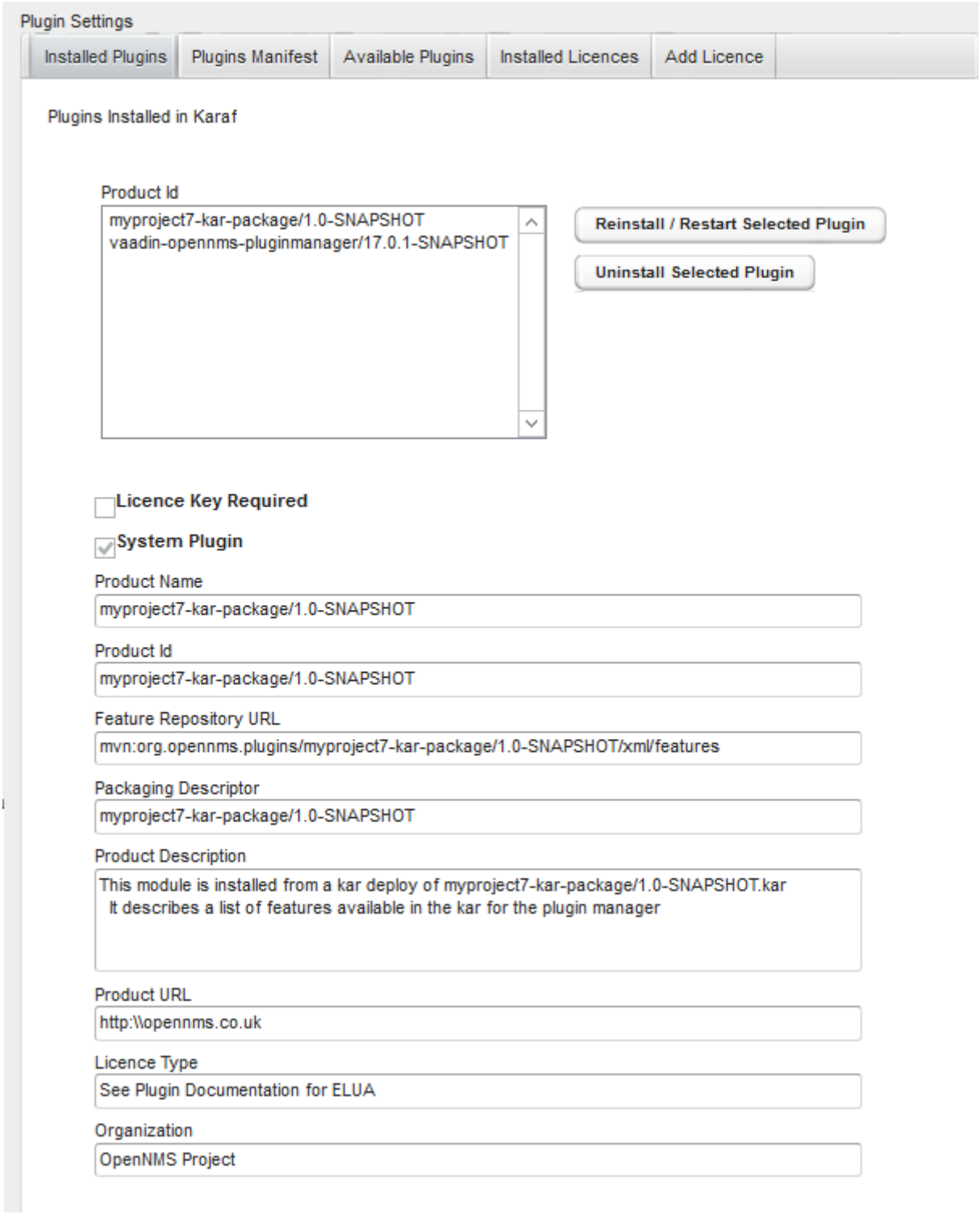


14.4. Installed Plugins

Under plugin settings, the Installed Plugins tab lists which plugins are currently installed in the *Karaf* instance selected in the *Karaf* instance data panel. System Plugins cannot be uninstalled through the UI. (The *Plugin Manager* is itself a system plugin). Non-system plugins can be reinstalled or removed from the system. Each plugin has metadata associated with it which is used to identify and describe the plugin.

Table 90. Plugin Metadata Fields

Plugin Metadata	Description
Product ID	The unique key used to identify the name and version of the feature. (Same as *Karaf* Feature Name/Version)
Licence Key Required	If true (ticked), this plugin needs a licence key to start
Licence Validated	If a licence key is required, a green text label will indicate if the licence has been installed and validated. Otherwise a red text label will indicate an invalid licence
System Plugin	If true (ticked) this is a system plugin and cannot be removed.
Packaging Descriptor	This describes the packaging mechanism by which the feature was delivered. This will refer to a Kar if the feature was manually installed as a Kar/RPM on the host server.
Feature Repository URL	The URL identifying the feature repository (Same as *Karaf* Feature Repository URL)
Product Description	A textual description of the functionality provided by the plugin.
Product URL	A URL to point to the plugin's documentation / web site
licence Type	A description of the licence applied to the plugin (May be GPL if the plugin is not subject to an ELUA)
Organisation	The organisation issuing the plugin and/or licence.



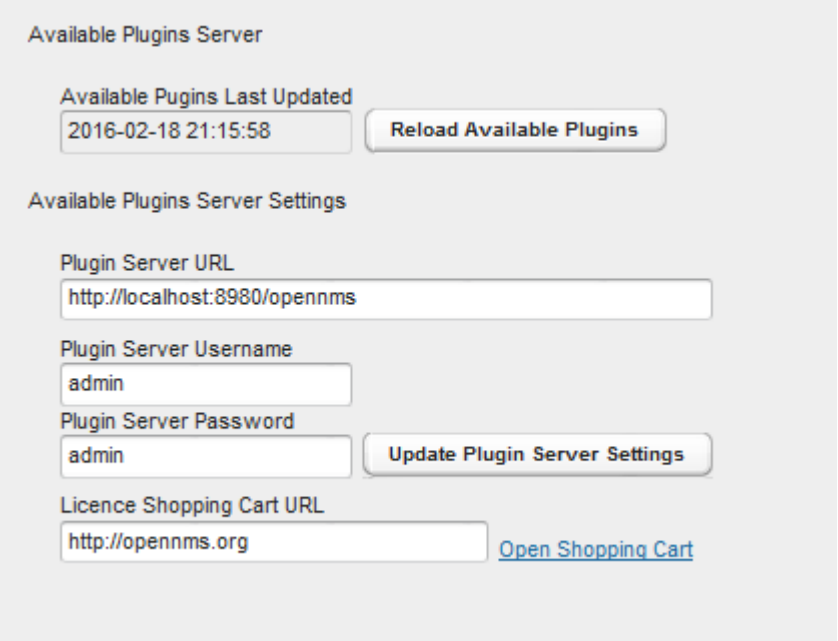
14.5. Available Plugins Server

The *Plugin Manager* obtains a list of available plugins from the 'Available Plugin's server'. This may also host a maven repo used to download the features if feature download from maven is enabled in *Karaf*.

Alternatively the *Plugin Manager* can list the available plugins which have been installed as bundled Plugin Kar/RPM's on the local machine. In this case, the *Plugin Server URL* should be pointed at the localhost.

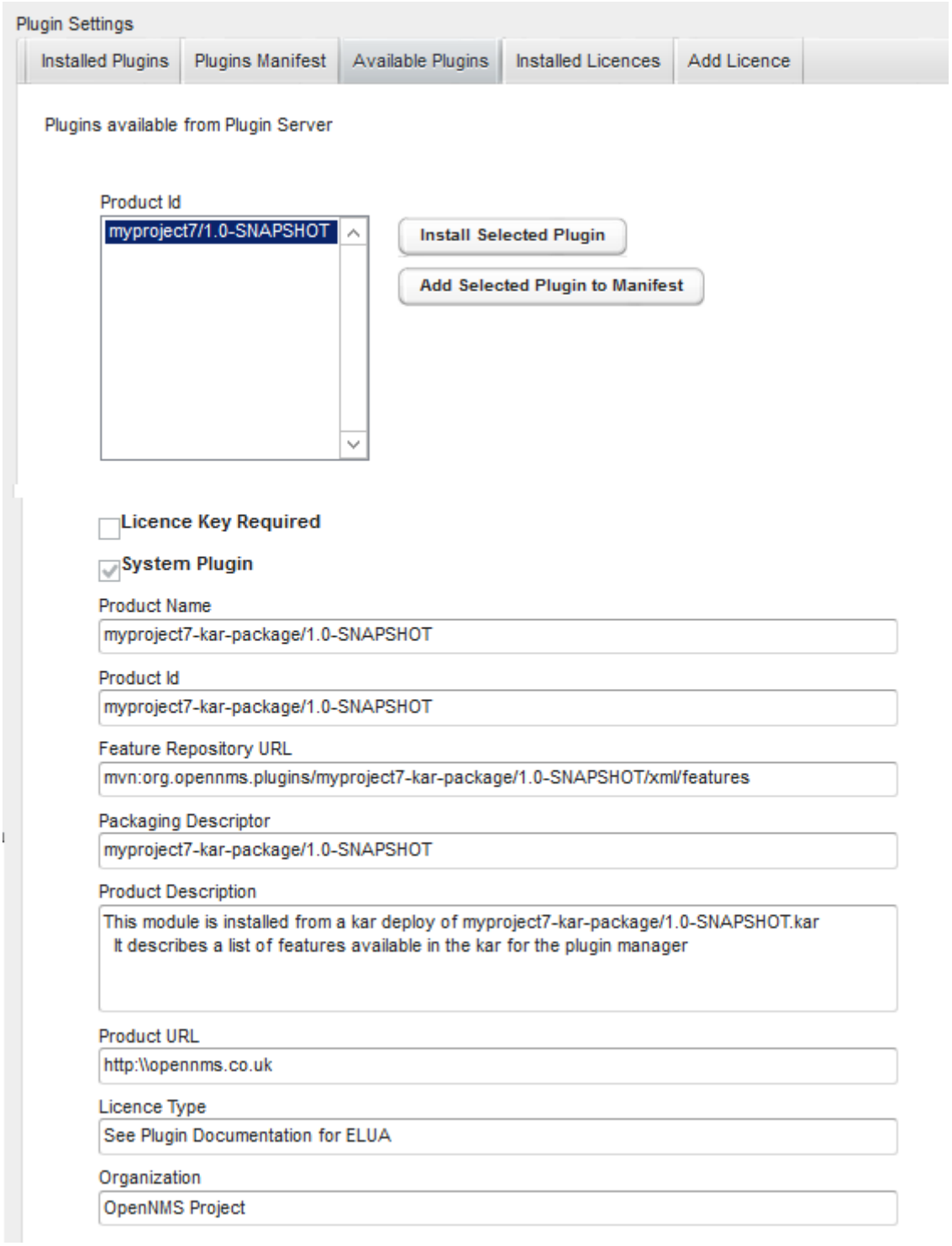
The admin username and passwords are used to access the *Available Plugins Server*. If a shopping

cart is provided for obtaining licences, the URL of the shopping cart should be filled in.



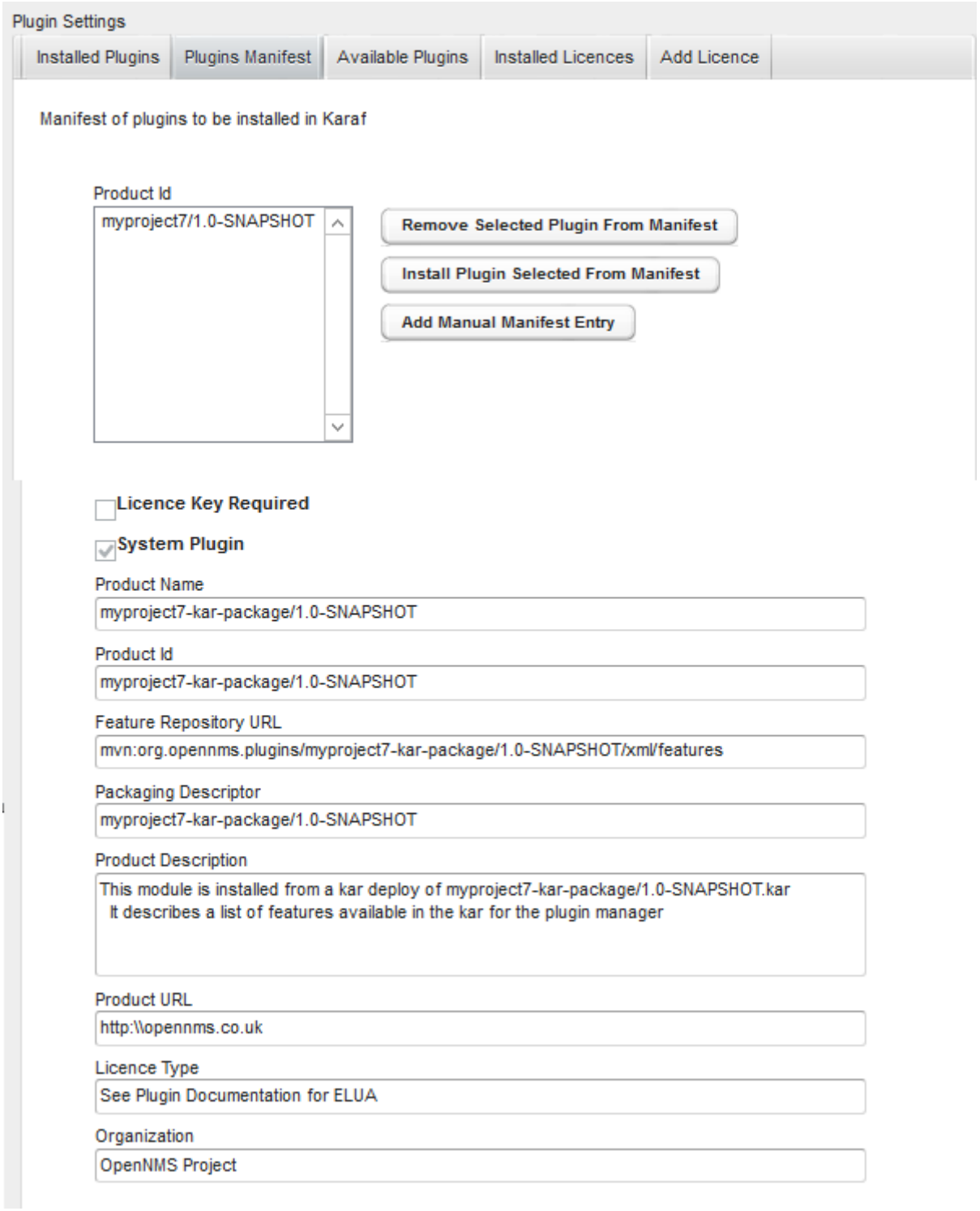
14.6. Installing Available Plugins

The Available Plugins panel list the plugins which are available and listed by the Available Plugins server. These can be directly installed into the selected *Karaf* instance or can be posted to a manifest for later installation. If a plugin is installed, the system will try and start it. However if a corresponding licence is required and not installed, the features will be loaded but not started. You must restart the feature if you later install a licence key.



14.7. Plugins Manifest

The Plugins Manifest for a given *Karaf* instance lists the target plugins which the *Karaf* instance should install when it next contacts the licence manager. If the *Plugin Manager* can communicate with the remote server, then a manifest can be selected for installation. A manual manifest entry can also be created for a feature. This can be used to install features which are not listed in the Available Features list.



14.8. Installed Licences Panel

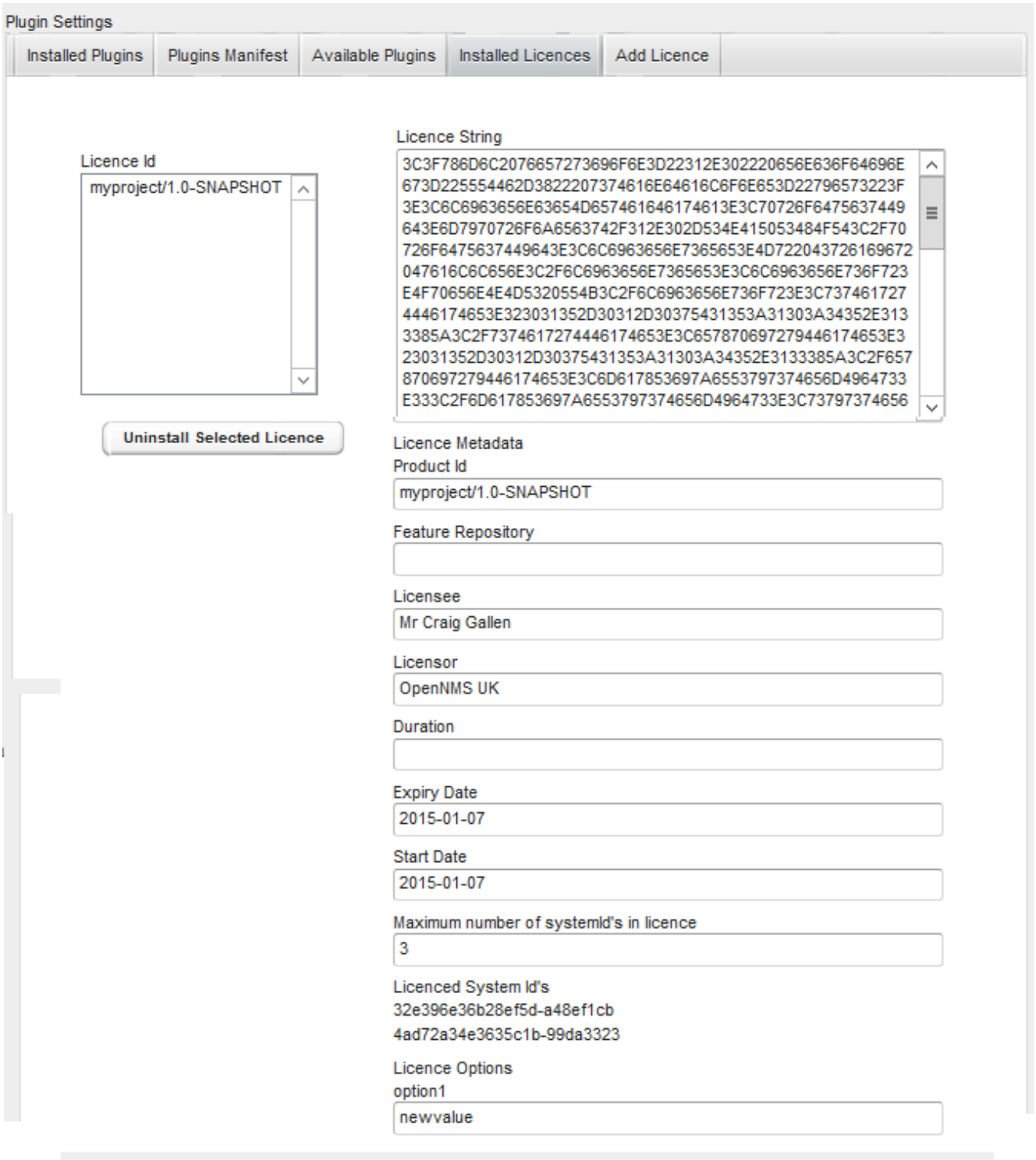
Each licence has a licence ID which is the *Karaf* feature ID of the feature to which the licence refers. Many licences can be installed on a system but only one licence string is allowed per feature ID.

Licence Strings are used to validate that a particular feature can be run on a given *Karaf* instance. The *Plugin Manager* will not allow a feature to run if it's licence cannot be validated using a private key encoded in the feature bundle.

Licences are associated with specific Product ID's and specific *Karaf* instances. Several *Karaf* instances can be listed in a licence allowing a feature to run on more than one system using the

same licence. When a licence is installed, the licence metadata is decoded and displayed.

A licence may be installed before or after its associated feature is installed. If a licence is installed after the feature the feature must be restarted before the licence will be read.



14.9. Adding a New Licence

New licences are added using the add licence panel. Licences are obtained from the *App Store* where they can be generated by a user for a given set of system id's.

A licence must be copied (cut and paste) from the app store into the add licence panel. The 'Validate licence' button should be used to check the licence has been installed correctly. Please note that this just checks the integrity of the licence string. A licence is only authenticated once it is installed and the corresponding feature bundle checks it on start-up.