Collecting integration data

OnCommand Insight

NetApp
October 04, 2024

Table of Contents

Collecting integration data

You can import integration data into your OnCommand insight system. Data can be imported using collectd, open source software that runs as a daemon to collect performance data, or by using the integration SNMP data source which allows you to collect generic SNMP data.

Data flow for integration data

The following applies to the total amount of integration data that is allowed to be presented to the OnCommand Insight server:

- A queue of 100 calls is maintained.

  When a client waits in the queue for more than one minute, a timeout error occurs.

- The recommended ingestion rate for integration data is once per minute, per client.
- There is a limit of 300 integration object types allowed.

Accessing collectd software and documentation

You can access the output writer plugin software and documentation for collectd at NetApp's GitHub site: https://github.com/NetApp/OCI_collectd

Backup and restore of integration data

Backup and restore of integration data is modeled after OnCommand Insight performance data backup and restore policies. When a backup is configured for performance data, integration data is also included in the backup. As with performance backup, the most recent seven days of integration data is included in the backup. Any integration data that is present in a backup is restored in a restore operation.

Licenses

A Perform license is required for integration data to be reported. If a Perform license is not present an error occurs with a message "Perform license required to report integration data".

Collecting SNMP Integration data

The integration SNMP data source allows you to collect generic SNMP data in OnCommand Insight.

Integration packs

The SNMP Integration data source uses an "Integration Pack" to define what integration values are collected, and what SNMP objects provide those values.

An Integration Pack consists of:

- A JSON configuration file (integration.json) defining integration payload contents in terms of SNMP objects of a specific device type (switch, router, and so on).
- A list of MIB files that the integration pack depends on.

An integration pack can define multiple data types. For example, when integrating an RHEL host, a data type can be defined for the general system information such as uptime, number of users, and number of running processes, a second data type can be defined for data on memory and file system usage. In general, each data type must be “flat” and cannot contain nested data.

A single integration pack should not define more than 24 data types. Insight limits the amount of integration data that is collected. Attempting to ingest more than 24 reports over a period of one minute results in a rate error.

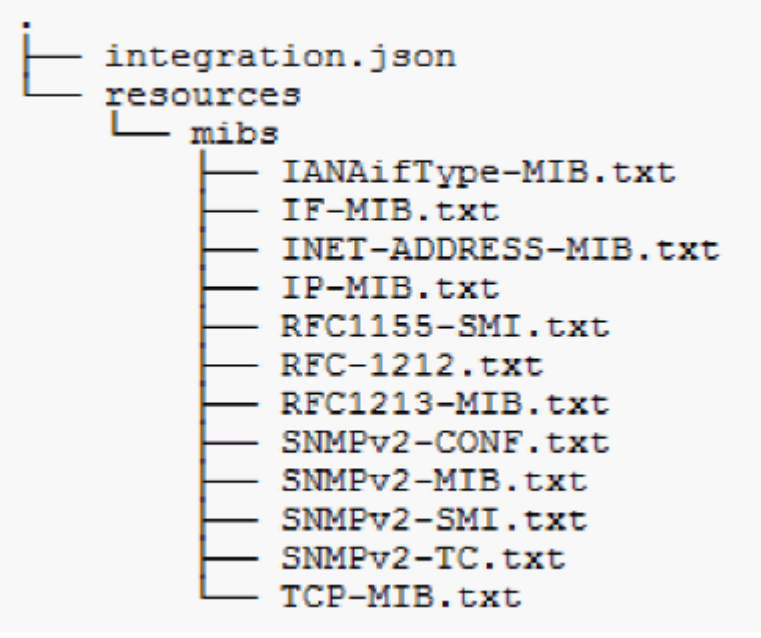
The names for integration types must adhere to the following rules:

- The name cannot start with the following characters: _, -, or,+
- The name cannot contain the following characters: #, \, /, *, ?, ", <, >, |, ' , `,
- Cannot be longer than 100 UTF-8 encoded bytes
- Cannot be named . or ..

Integration file format

An integration pack is a ZIP file that contains a JSON configuration file (integration.json) defining the integration payload contents in terms of SNMP objects. It also contains a MIBS folder that contains all of the MIB files and their MIB dependencies.

The `integration.json` file must exist at the top level of the ZIP file and the MIB files must exist in the "resources/mibs" subdirectory within the ZIP. The ZIP file may also contain files, such as a "readme.txt", if desired. An example of integration ZIP structure is:



Importing SNMP integration packs

You import SNMP integration packs into OnCommand Insight using the web UI.

Integration packs are identified by the "integrationPackName" value defined in the `integration.json` configuration file contained in the ZIP file.

Before you begin

You must have created a properly formatted ZIP file that contains the integration pack you want to import to the OnCommand Insight server.

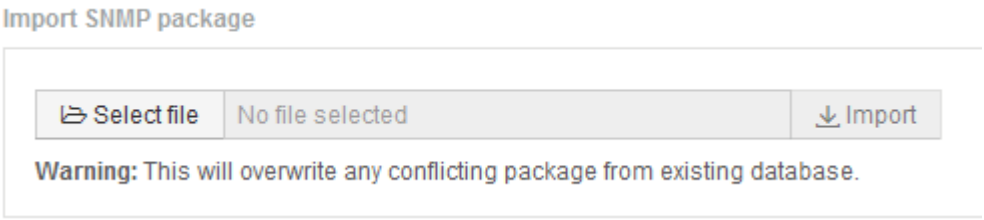
About this task

Use the following steps to import SNMP integration packs to the Insight server.

Steps

1. Click **Admin** > **Setup** > **SNMP Integration**

   The system displays the Import SNMP package screen:

2. Click **Select file** to select the local file containing the SNMP package.

   The file you select is displayed in the File box.

   Any existing integration pack with the same name is overwritten.

3. Click **Import**

   The file is imported to the Insight server.

Creating an SNMP integration data source

The Integration SNMP Data Source provides common SNMP configuration properties similar to other SNMP based data sources included with the OnCommand Insight data sources for Brocade and Cisco.

Before you begin

In order to successfully use the Integration SNMP data source to collect, the following must be true:

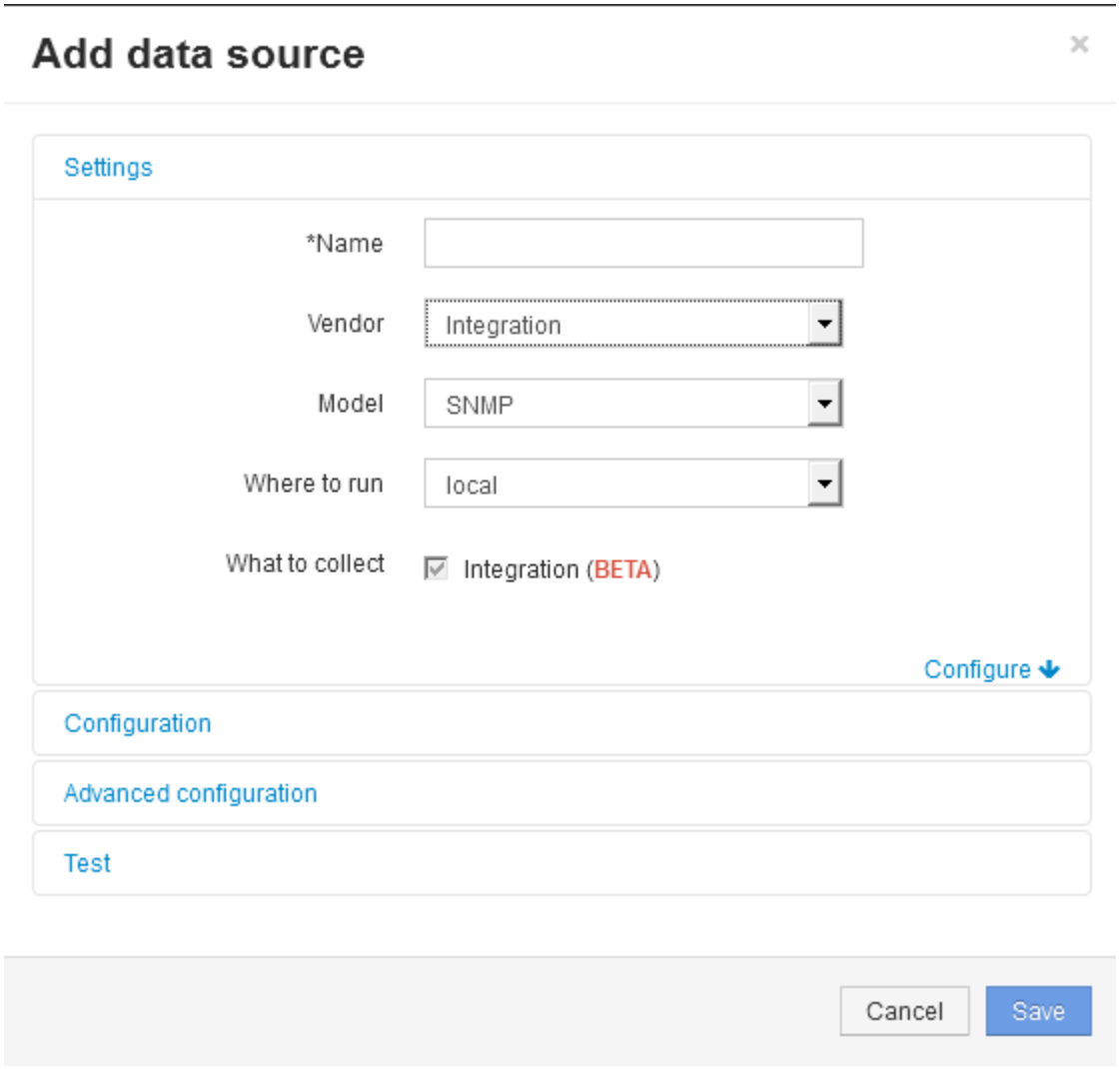
- You must have already imported an integration pack you will use for this SNMP data source.
- All target devices share the same credentials.
- All target devices implement the SNMP Objects referenced by the configured integration pack.

About this task

To create an SNMP Integration data source, choose vendor "Integration" and model "SNMP" in the data source creation wizard.

Steps

1. In the OnCommand Insight web UI, click **Admin** > **Data Sources**
2. Click **+Add**
3. Enter a name for the Data Source
4. For Vendor, select **Integration**
5. For Model, select **SNMP**



6. For What to collect, check **Integration**

   This is the only package on this data source and is checked by default:

7. Click **Configuration**
8. Enter the IP addresses for the systems from which you will collect SNMP data
9. Select an imported SNMP Integration Pack

10. Set the integration poll interval
11. Select the SNMP version
12. Enter the SNMP community string

    For SNMP V1 and V2.

13. Add the user name and password for systems you will be collecting data from.

    For SNMP V3.

14. Click **Advanced Configuration**

    The Advanced Configuration default settings are displayed. Make any changes to these settings that are required.

Integration.json file information

The integration.json file identifies the payload .

The following illustration provides a color-coded representation of a simple integration.json file. The accompanying table identifies the function of the objects in the file.

```
{
 "integrationPackName": "WindowsSnmp",
 "description": "Generic integration for mibs supported by the default
SNMP  Agent for Windows 2012, including HOST-RESOURCES",
 "acquisitionType": "SNMP",
 "integrationTypes": [
  {
   "integrationType":       "snmp_win2012_host",
   "name": {
    "mibModuleName":     "RFC1213-MIB",
    "objectName":        "sysName"
   },
   "identifiers": {
    "hostname": {
     "mibModuleName":     "RFC1213-MIB",
    }
   },
   "attributes": {
    "description": {
     "mibModuleName":     "RFC1213-MIB",
     "objectName":        "sysDescr"
    },
    "snmp_sys_obj_id": {
     "mibModuleName":     "RFC1213-MIB",
     "objectName":        "sysObjectID"
    }
   },
   "dataPoints": {
    "uptime": {
     "num": {
      "mibModuleName":    "RFC1213-MIB",
      "objectName":       "sysUpTime"
     }
    }
   }
  }]}
```

Blue	Reserved
Red	User customizable strings and IDs
Green	MIB names
Purple	MIB object
Black	JSON structure

About integration.json files

Each field has the following characteristics:

- The "identifiers" section forms a unique compound key to create a new "object" in Insight
- The "attributes" provide supporting meta-data about the object.

  In both of these cases, only the value of the latest report for that object (identified by the identifiers) is preserved.

- The "dataPoints" are time-series data and must be numeric values. Insight keeps each and every value reported here for 90 days (by default) and links them time-series to the object identified.

Numeric Expressions

By default, all value expressions are reported as strings in the integration payload. "identifiers" and "attributes" may only define string values. "dataPoints" may define string or numeric values. Numeric values are defined using one of the following modifier keys:

- num - the total number of bytes received since the counter was last initialized
- delta - the number of bytes received during the poll interval
- rate - the average receive rate during the poll interval in bytes per second

  An average receive rate during the poll interval in megabytes per second can be accomplished using a combination of rate and math operations

Math operations

The `integration.json` file supports the following math operations: add, subtract, multiply, divide. The following example shows multiplication, division, and sum operations in a JSON file.

```
"network_utilization":
{
  "mult": [
    {
      "div": [
        {
          "sum": [
            "rate": {
              "mibModuleName":   "IF-MIB",
              "objectName":      "ifHCOutOctets",
              "comment":         "bytes per second out"
            },
            "rate": {
              "mibModuleName":   "IF-MIB",
              "objectName":      "ifHCInOctets",
              "comment":         "bytes per second in"
            }
          ]
        },
        {
          "num": {
            "mibModuleName":   "IF-MIB",
            "objectName":      "ifSpeed",
            "comment":         "1,000,000 bits per second"
          }
        }
      ]
    },
    {
      "const":                  0.0008,
      "comment":        "normalize to ratio of bits and convert to percent:
                        8 * 100 / 1,000,000 = 0.0008"
    }
  ]
}
```

Keywords

An integration pack keyword, string, is implemented to force OCTET STRINGs or proprietary types derived from OCTET STRING that would normally be rendered in hexadecimal format to instead be rendered as ASCII characters.

Often OCTET STRINGs contain binary data, for example MAC addresses and WWNs:

```
        "interface_mac": {
                  "mibModuleName":       "IF-MIB",
                  "objectName":          "ifPhysAddress"
                 }
```

ifPhysAddress is type PhysAddress, which is just an OCTET STRING:

```
  PhysAddress ::= TEXTUAL-CONVENTION
                DISPLAY-HINT "1x:"
                STATUS       current
                DESCRIPTION
                                "Represents media- or physical-level
addresses."
                SYNTAX       OCTET STRING
```

When ifPhysAddress is rendered as hex by default, the result is:

```
"interface_mac": "00:50:56:A2:07:E7"
```

However if you have an OCTET STRING or proprietary type derived from OCTET STRING that you want to interpret as ASCII, you can use the "string" keyword:

```
        "string_test_1": {
          "string": {
            "mibModuleName":      "IF-MIB",
            "objectName":         "ifPhysAddress"
          }
        },

        "string_test_2": {
          "string": [
            {
              "mibModuleName":      "IF-MIB",
              "objectName":         "ifPhysAddress"
            },
            {
              "const": "JSD"
            },
            {
              "mibModuleName":      "IF-MIB",
              "objectName":         "ifPhysAddress"
            }
          ]
        }
```

The keyword follows the existing string concatenation rules, inserting a single space between terms in the following example:

```
    "string_test_1": "PV¢ç",
            "string_test_2": "PV¢ç JSD PV¢ç"
```

The "string" keyword acts on a single term or a list of terms, but not nested expressions. Nested expressions are only supported for dataPoint expressions. Attempting to use a "string" expression in a dataPoint expression will result in an error similar to the following:

java.lang.IllegalArgumentException: Integration pack 'GenericSwitch32' index 'snmp_generic_interface_32' section 'dataPoints' key 'string_test_3' unsupported JSON numeric expression '{"string":{"mibModuleName":"IF-MIB","objectName":"ifPhysAddress"}}'

Some derived OCTET STRING types such as DisplayString, SnmpAdminString have hard-coded precedence over the "string" keyword. This is because SnmpAdminString is specifically UTF-8 encoded, and we want to handle it correctly, whereas the "string" keyword forces the default string representation returned by the snmp_framework, which assumes single byte ascii code points per character.

Copyright information

Trademark information