cisco-gNMI-main

June 19, 2021

0.1 Understand gNMI and how to build your first gNMI client with Python to interwork with IOS-XR

The following practical examples address the capabilities of IOS XR to communicate over gNMI (gRPC Network Management Interface). These capabilities are tightly tied to the specifications of the gNMI service. We will explore the gNMI capabilities for automated configuration of the IOS XR device and monitor in real-time the data rate of an interface of the device.

All Cisco platforms currently support the gNMI protocol. In the case of IOS XR, the protocol is supported beginning with release 6.5.1:

gNMI is an alternative to network management protocols such as NETCONF and RESTCONF and it is built in the gRPC Remote Procedure Call (gRPC) framework. Therefore, in order to understand gNMI we first need to understand gRPC.

NOTE

References on gRPC and gNMI: - gRPC - gNMI Specification - Draft - Openconfig Rtgwg gNMI Spec - v01

References on programmability of IOS XR:

ASR 9000 - Programmability Configuration Guide - Yang Data Models - Telemetry Configuration

NCS 5000 - Programmability Configuration Guide - YANG Data Models - Telemetry Configuration

gRPC overview gRPC is a framework dedicated for connecting services and it offers support for load-balancing, tracing, authentication and health-checking. For example, we could use gRPC to connect front-end applications with backend services or to connect management systems with network elements.

In gRPC, we define a service using protocol buffers (protobuf).

gRPC transport gRPC is based on the HTTP/2 transport protocol and it has support for Remote Procedure Calls (RPCs). These are operations that can be called remotely with a set of given parameters and they can return information back to the requestor. The following categories of RPCs exist: - Unary RPC - the client sends a single request to the server and receives a single response back (e.g. polling information). - Server streaming RPC - the client sends a request to get multiple streams of information from the server (e.g. model-driven telemetry); the client can wait for the completion marked in the metadata by the server. - Client streaming RPC - the client sends multiple streams of information to the server; the server typically sends a single response back - Bidirectional streaming RPC - streaming of multiple read-write streams between the client and the server.

gRPC implementation The communication between the server and the client is established through a gRPC channel formed between the hosts. Therefore, we require the specification of the IP address and the port of the host for the server. A client could modify the traits of a channel (enable / disable compression of a message) and could monitor the state of the channel.

The server side contains the implementation of the abstract service and will handle RPC calls and encode the responses. The client side contains the implementation of the same service and can call the methods on the gRPC channel established between the client and the server.

gRPC authentication In the context of network management, gRPC supports SSL/TLS authentication. It is also possible to add custom authentication methods by extending the implementation.

The client uses SSL/TLS authentication to authenticate to the server and to encrypt the messages. There is also an option to use certificates for authentication. The SSL credentials are attached to the gRPC channel.

Protobuf The payload message transferred through the RPCs may be encoded in protobuf. Note that it is also possible to use alternative encoding formats, such as JSON.

Protobuf is a form of binary serialization with which one can encode an abstract concept (such as a service) in a structured way. The format is similar to XML format but said to be simpler, smaller and hence more efficient. The extensions of a service are backwards-compatible with older versions of it.

Given a protobuf file, one can generate bindings in various programming languages for the implementation of the underlying service described in the proto file. Proto files usually also have data structures defined in them which describe messages that are transferred with the RPC calls. Due to this definition, we can avoid hand-parsing the messages that are passed along. All these mean that we can transition from an abstract concept and automatize the utilization of that concept.

NOTE - For a reference on the gNMI service definition, see its corresponding protobuf definition at: https://github.com/openconfig/gnmi/blob/master/proto/gnmi/gnmi.proto.

gNMI gNMI is dedicated for managing and monitoring the state of network elements. It supports: - Configuration management and retrieval - Operational state retrieval - Bulk data collection via Model-Driven Telemetry (MDT)

In practice, the configuration is performed by the management system (client) that has read-write permissions and is applied to the network element (target).

The messages within the gRPC service are defined as protocol buffers. The data has to be structured in a tree-like structure, similar to YANG models, and it has to be addressable by paths within that tree.

gNMI data payloads and paths In general, gNMI carries instances of YANG schemas but it can be used to carry data of other types of structures. In any case, a node will be identified by a path within the structure's tree and its value will be serialized.

gNMI RPCs The gNMI RPCs sent between the client and the server are: - Capabilities RPC - learn about the features supported by the network element. - Subscribe RPC - a bidirectional streaming RPC; the data can be sent as soon as it is available and hence avoids overloading the network element. - Set RPC - a unary RPC; it is sent by the client to update the state of the target (update / delete / replace). - Get RPC - a unary RPC; it is sent by the client to ask a snapshot of the target's state (configuration / operational).

This notebook will address four possible RPC calls that can be made with gNMI: - Retrieval of capabilities of a network element - Retrieval of state based on CLI format - Retrieval and configuration of state based on YANG and XPaths - Configuration of MDT streaming

0.1.1 Preparation of the environment

TASK 0 (OPTIONAL)

Enable logging at the debug level:

[1]:
```python
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
import logging
logger = logging.getLogger()
logger.setLevel(logging.DEBUG)
logging.debug("test")
```

```
DEBUG:root:test
```

There are several implementations of gNMI clients in the open-source community in various programming languages (Go, Python, C++, Java). In this laboratory, we will use the `cisco-gnmi` Python implementation for the communication with the gNMI server hosted on the IOS XR device.

NOTE - For more information about the `cisco-gnmi` Python implementation see https://github.com/cisco-ie/cisco-gnmi-python.

TASK 1

Import the Cisco gNMI package and its client module:

[2]:
```python
from cisco_gnmi import ClientBuilder
```

0.1.2 Connectivity to the device

TASK 2

Assuming that the device has its gRPC interface configured (`grpc port 57777`), connect to the device using the following details: - IP address: `198.18.134.72` - gRPC port: `57777` - operating system: `IOS XR` - username: `cisco` - password: `cisco`

The `ClientBuilder()` instantiates a builder for the gNMI client with the IP address and the port of the gNMI target (IOS XR device). By default, the target will expect secure authentication over TLS. The connecting element (the Jupyter host) will retrieve the certificate from the target using the `set_secure_from_target()` and will derive the SSL target name from this certificate using the `set_ssl_target_override()` method. Finally, the client will authenticate with the username and password of the target using the `set_call_authentication()` method. We assume that the user has read-write permissions.

Finally, we construct the client by calling `construct()`.

[7]:
```
# The way this works:
# 1. Assume gRPC over TLS is configured on the IOS-XR device (grpc port 57777)
# 2. Retrieve the certificate from the IOS-XR
# 3. Derive the SSL target name from the certificate
# 4. Authenticate

builder = ClientBuilder('10.58.50.234:57000')
#builder = ClientBuilder('198.18.134.72:57777')
builder.set_os('IOS XR')
builder.set_secure_from_target()
builder.set_ssl_target_override()
builder.set_call_authentication('cisco', 'cisco123')
#builder.set_call_authentication('cisco', 'cisco')

client = builder.construct()
```

```
DEBUG:cisco_gnmi.builder:Using IOS XR wrapper.
```

[7]:
```
<cisco_gnmi.builder.ClientBuilder at 0x10d6e5df0>
```

[7]:
```
<cisco_gnmi.builder.ClientBuilder at 0x10d6e5df0>
```

[7]:
```
<cisco_gnmi.builder.ClientBuilder at 0x10d6e5df0>
```

[7]:
```
<cisco_gnmi.builder.ClientBuilder at 0x10d6e5df0>
```

```
DEBUG:cisco_gnmi.builder:Using secure channel.
DEBUG:cisco_gnmi.builder:Using username/password call authentication.
DEBUG:cisco_gnmi.builder:Using SSL/metadata authentication composite
credentials.
DEBUG:cisco_gnmi.util:Using ems.cisco.com as certificate CN.
WARNING:cisco_gnmi.builder:Overriding SSL option from certificate could increase
MITM susceptibility!
```

0.1.3 Retrieval of gNMI capabilities

A client can discover the capabilities of its target using the `Capabilities` RPC. The target will reply with a message that contains: - the version of its gNMI service - the supported encodings - the supported data models (YANG models)

TASK 3

Request the capabilities of the IOS XR target:

[ ]:
```
capabilities = client.capabilities()
print(capabilities)
```

0.1.4 Retrieval of state information based on CLI format

The CLI format is, understandably, the most widely distributed method for device configuration. Despite its readability for humans, it is not the most suitable for computer programs that can now automatize steps of our state retrieval and configuration. The reason behind this is the same as for parsing logs, the CLI configuration content is semi-structured. This makes the development of automation applications laborious with potentially an unknown number of special use-cases, depending on how the commands are defined in their implementation at the source, i.e., in the IOS XR operating system.

Nevertheless, practicing an already existing skill (CLI configuration) within a new context (over gNMI) can help to isolate and to understand the nature of gNMI. We will start by executing a CLI command over gNMI.

For reasons of simplification and to avoid duplication of parts of code that we have for running and printing the outcome of a CLI command, we can define two methods that incorporates these two steps.

The `get_cli()` wrapper accepts two parameters: an instance of the gNMI client and a command in string format. Inside this function, we will make a call to the `get_cli()` function of the client. Internally, this function will send a `Get` RPC request on behalf of the client with the following parameters: - encoding: ASCII - the path for which the data should be retrieved: CLI command

TASK 4

Define the following two methods:

[8]:
```
def print_context(reply):
    print("timestamp:", reply.notification[0].timestamp)
    print("path:", reply.notification[0].update[0].path.elem[0].name)
```

[9]:
```
def get_cli(client, command):
    get_reply = client.get_cli(command)

    print_context(get_reply)
    print(get_reply.notification[0].update[0].val.ascii_val)
```

TASK 5

Firstly, let's retrieve the entire running configuration from the device. Notice that because the CLI command is sent in ASCII format, the device responds to the request with its running **configuration** in the same format:

[ ]:
```
command = 'show running-config'
get_cli(client, command)
```

TASK 6

We can also retrieve a subsection of the running configuration. Let's retrieve the **configuration state** for streaming of model-driven telemetry:

```
[10]: command = 'show running-config telemetry model-driven'
      get_cli(client, command)
```

```
DEBUG:cisco_gnmi.client:path {
  elem {
    name: "show running-config telemetry model-driven"
  }
}
encoding: ASCII

timestamp: 1624086168824873924
path: show running-config telemetry model-driven

----------------- show running-config telemetry model-driven ------------------
% No such configuration item(s)
```

TASK 7

Correspondingly, we can also retrieve information about the operational state of the device by running the corresponding command. Let's retrieve the **operational state** of streaming of model-driven telemetry:

```
[11]: command = "show telemetry model-driven subscription"
      get_cli(client, command)
```

```
DEBUG:cisco_gnmi.client:path {
  elem {
    name: "show telemetry model-driven subscription"
  }
}
encoding: ASCII

timestamp: 1624086175619052122
path: show telemetry model-driven subscription

------------------ show telemetry model-driven subscription -------------------
```

Notice that the last two commands did not retrieve any information. This is because telemetry streaming is not *yet* configured or active on the IOS XR device. In the next tasks, we will learn how to configure telemetry streaming and how to receive the streamed information.

TASK 8

Inspect the current gRPC configuration on the device:

[12]:
```
command = "show running-config grpc"
get_cli(client, command)
```

```
DEBUG:cisco_gnmi.client:path {
  elem {
    name: "show running-config grpc"
  }
}
encoding: ASCII

timestamp: 1624086183883178953
path: show running-config grpc

-------------------------- show running-config grpc ---------------------------
grpc
 port 57000
!
```

0.1.5 Retrieval and configuration of state information based on YANG and XPaths

NOTE - References on YANG and JSON encoding for YANG: - RFC 6020 - YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF) - RFC 7950 - The YANG 1.1 Data Modeling Language - RFC 7951 - JSON Encoding of Data Modeled with YANG

YANG - A Data Modeling Language YANG is a modelling language designed for configuration and description of the state of a system. Its purpose is to enable the manipulation of a system using the Network Configuration Protocol (NETCONF) with NETCONF Remote Procedure Calls (RPCs) and Notifications.

YANG models are defined in highly structured text files and are compilable. YANG's statically-typed syntax and semantic definition ensures the verification of the model's definition at compile time, resulting in errors that can be treated early in the development process.

The models have a tree-based structure with nested sub-elements, named schema nodes. Although they are human readable, the strict formatting and the RFCs defined around them ensure reliability for processing and automation of instance creation of these models.

A network device is said to support a capability if it contains a data model definition of that capability in the form of a module. This module will then contain definitions of data nodes and may import or include definitions from other modules. Such a module is called self-contained and compilable.

A data model can be instantiated in the Extensible Markup Language (XML) or in Javascript Object Notation (JSON).

Snippet of a YANG model

```
module Cisco-IOS-XR-telemetry-model-driven-cfg {

  /*** NAMESPACE / PREFIX DEFINITION ***/

  namespace "http://cisco.com/ns/yang"+
    "/Cisco-IOS-XR-telemetry-model-driven-cfg";


  prefix "telemetry-model-driven-cfg";

  /*** LINKAGE (IMPORTS / INCLUDES) ***/

  import ietf-inet-types { prefix "inet"; }

  import Cisco-IOS-XR-types { prefix "xr"; }

  import cisco-semver { prefix "semver"; }

  [...]

  container telemetry-model-driven {
    status "deprecated";
    description
      "This model is deprecated and is replaced by
      Cisco-IOS-XR-um-telemetry-model-driven-cfg.yang
      which will provide the compatible functionalities
      .  Model Driven Telemetry configuration";

    container sensor-groups {
      description "Sensor group configuration";

      list sensor-group {
        key "sensor-group-identifier";
        description "Sensor group configuration";

        container sensor-paths {
          description "Sensor path configuration";

          list sensor-path {
            key "telemetry-sensor-path";
            description "Sensor path configuration";
            leaf telemetry-sensor-path {
              type string;
              description "Sensor Path";
            }
          }
        }
        leaf sensor-group-identifier {
          type xr:Cisco-ios-xr-string;
          description "The identifier for this group";
        }
      }
    }
```

Snippet of a YANG instance encoded in XML

```
<telemetry-model-driven xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-telemetry-model-driven-cf
   <sensor-groups>
    <sensor-group>
     <sensor-group-identifier>S1</sensor-group-identifier>
     <sensor-paths>
      <sensor-path>
       <telemetry-sensor-path>Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/inter
      </sensor-path>
     </sensor-paths>
    </sensor-group>
   </sensor-groups>
</telemetry-model-driven>
```

Snippet of a YANG instance encoded in JSON

```
{
  "Cisco-IOS-XR-telemetry-model-driven-cfg:telemetry-model-driven":  {
     "sensor-groups": {
       "sensor-group": [{
          "sensor-group-identifier": "S1",
            "sensor-paths": [{
               "sensor-path": {
                   "telemetry-sensor-path":"Cisco-IOS-XR-infra-statsd-oper:infra-statistics/in
               }
            }]
        }]
     }
}
```

YANG compatibility with the gNMI protocol The gRPC Remote Procedure Call protocol is a newer protocol than NETCONF. It is the result of the adoption of microservices in new architectures. The switch from monolithic applications to ones with distributed components has encouraged the development of the gRPC, and later, of the gNMI protocols. These protocols allow for frequent and efficient RPCs in the microservices systems and they work over HTTP/2.

Similarly to NETCONF, their purpose is to manage and retrieve the configuration of a system, as well as to stream telemetry data about that system. A gNMI service will carry payloads that are instances of YANG data models in its attempt to manage or retrieve the underlying information.

The values for each instance of the data model are serialized using JSON, Protobuf, Bytes, ASCII or JSON_IETF encoding.

Each gNMI request will have a Path defined in a structured format that, optionally, contains keys. A Path contains: - **origin**: optional; its presence disambiguates the path reference (e.g., specifying a namespace) - **elem**: array of PathElem; a PathElem is an instance of a node from the data tree and is composed of its name and its keys and attributes - **target**: optional; its presence is marked in the Prefix of a request and it labels a stream of data for the client that does the request

Here is an example of an encoded Path that corresponds to the node `Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface[interface-name="MgmtEth0/I`

```
path: <
  origin: Cisco-IOS-XR-infra-statsd-oper
  elem: <
    name: "infra-statistics"
  >
  elem: <
    name: "interfaces"
  >
  elem: <
    name: "interface"
    key: <
      key: "interface-name"
      value: "MgmtEth0/RP0/CPU0/0"
    >
  >
  elem: <
    name: "generic-counters"
  >
>
```

NOTE - For a reference on the gNMI service specification for the Path component, see its corresponding protobuf definition at: https://github.com/openconfig/gnmi/blob/master/proto/gnmi/gnmi.proto#L129.

XPath The XML Path Language (XPath) is a language for referencing components in an XML document. Specifically, the most important type in XPath is the LocationPath type that will describe in an expression the absolute or relative location of a node within the hierarchy of the XML tree.

XPath can be used with YANG to address nodes in both the definition and the instantiation of data models. It can further be used in NETCONF and gNMI requests to reference a node or to reference and filter the retrieved data.

NOTE - References: - The XPath standards - Whitepaper - XPath in NETCONF and YANG

TASK 9

For reasons of simplification, define a method that pretty-prints JSON content (`print_json()`).

Furthermore, define another method that makes the Get request and displays the reply (get_xpath()). Internally, this function will send a Get RPC request on behalf of the client with the following parameters: - encoding: JSON_IETF - the path for which the data should be retrieved: xpath

[13]:
```
import json

def print_json(reply):
    # Pretty printing the JSON reply
    json_reply = reply.notification[0].update[0].val.json_ietf_val
    if json_reply:
        val_dict = json.loads(json_reply)
        print(json.dumps(val_dict, indent=4, sort_keys=True))
```

[14]:
```
def get_xpath(client, xpath):
    get_reply = client.get_xpaths(xpath)

    print_context(get_reply)
    print_json(get_reply)
```

TASK 10

Use the XPath format to reference a YANG structure whose information we want to retrieve. Specifically, we will retrieve the interfaces configured on the IOS XR device. We specify this by first mentioning the **origin** (`Cisco-IOS-XR-ifmgr-cfg`), followed by `:`, and then followed by a container or other YANG structure that is part of the YANG model (here, `interface-configurations`):

[15]:
```
xpath = 'Cisco-IOS-XR-ifmgr-cfg:interface-configurations'
get_xpath(client, xpath)
```

```
DEBUG:cisco_gnmi.client:path {
  origin: "Cisco-IOS-XR-ifmgr-cfg"
  elem {
    name: "interface-configurations"
  }
}
encoding: JSON_IETF

timestamp: 1624086207340411282
path: interface-configurations
{
    "interface-configuration": [
        {
            "Cisco-IOS-XR-ipv4-io-cfg:ipv4-network": {
                "addresses": {
                    "primary": {
                        "address": "192.168.0.1",
                        "netmask": "255.255.255.255"
                    }
                }
            },
            "active": "act",
            "interface-name": "Loopback0",
            "interface-virtual": [
                null
            ]
        },
        {
            "active": "act",
            "interface-name": "MgmtEth0/RP0/CPU0/0",
            "shutdown": [
                null
            ]
        },
        {
            "Cisco-IOS-XR-cdp-cfg:cdp": {
                "enable": [
                    null
                ]
            },
            "Cisco-IOS-XR-ipv4-io-cfg:ipv4-network": {
                "addresses": {
                    "primary": {
                        "address": "172.16.0.1",
                        "netmask": "255.255.255.252"
                    }
                }
            },
            "active": "act",
            "description": "To xr9kv-2/GigE0/0/0/0",
            "interface-name": "GigabitEthernet0/0/0/0"
        },
        {
            "Cisco-IOS-XR-cdp-cfg:cdp": {
                "enable": [
                    null
                ]
            },
            "Cisco-IOS-XR-ipv4-io-cfg:ipv4-network": {
                "addresses": {
                    "primary": {
                        "address": "172.16.0.5",
                        "netmask": "255.255.255.252"
                    }
                }
            },
            "active": "act",
            "description": "To xr9kv-3/GigE0/0/0/1",
            "interface-name": "GigabitEthernet0/0/0/1"
        },
        {
            "Cisco-IOS-XR-cdp-cfg:cdp": {
                "enable": [
                    null
                ]
            },
            "Cisco-IOS-XR-ipv4-io-cfg:ipv4-network": {
                "addresses": {
                    "primary": {
                        "address": "172.16.0.9",
                        "netmask": "255.255.255.252"
                    }
                }
            },
            "active": "act",
            "description": "To xr9kv-5/GigE0/0/0/0",
            "interface-name": "GigabitEthernet0/0/0/2"
        },
        {
            "Cisco-IOS-XR-cdp-cfg:cdp": {
                "enable": [
                    null
                ]
            },
            "Cisco-IOS-XR-ipv4-io-cfg:ipv4-network": {
                "addresses": {
                    "primary": {
                        "address": "172.16.0.13",
                        "netmask": "255.255.255.252"
                    }
                }
            },
            "active": "act",
            "description": "To xr9kv-4/GigE0/0/0/0",
            "interface-name": "GigabitEthernet0/0/0/3"
        },
        {
            "Cisco-IOS-XR-ipv4-io-cfg:ipv4-network": {
                "addresses": {
                    "primary": {
                        "address": "10.58.50.234",
                        "netmask": "255.255.255.192"
                    }
                }
            },
            "active": "act",
            "interface-name": "GigabitEthernet0/0/0/4"
        }
    ]
}
```

0.1.6 Configuration of model-driven telemetry streaming (persistent, dial-out)

Configure the IOS XR device for model-driven telemetry streaming using the **gRPC dial-out** mechanism. In this case, the device initiates a gRPC session (**dial-out**) with the MDT receiver (i.e., the host that runs the Jupyter notebook - 198.18.134.50) and exchanges SYN – SYN-ACK – ACK with it while establishing the connection.

If the connection is successfully created, the device will start streaming telemetry data towards the collection point.

The streaming job will run indefinitely on the IOS XR device, even if the receiver is unreachable. The device will repeatedly try to contact the receiver, every 30 seconds.

NOTE - The dial-out mechanism is supported in both TCP and gRPC.

A dial-out MDT streaming configuration is **persistent** in the configuration of the IOS XR device and will be consistent throughout device reboots. It consists of: - A **sensor-group** that is composed of sensor-path(s): - A sensor-path is a specification of the YANG component about which information should be streamed; specified in XPath format - A **destination-group** that is composed of destination(s): - A destination contains the receiver's specifications: - IP address - port - encoding format - protocol - A **subscription** that brings together a sensor-group, a destination-group and a time interval [ms]

Here, we subscribe to updates of the operational state (`Cisco-IOS-XR-infra-statsd-oper`) of all the interfaces. Specifically, we are looking for their generic counters: - [multicast | broadcast] packets received | sent - bytes received | sent - output | input drops - output | input queue drops - CRC errors - …

In this example, we expect to receive data regularly, every 60 seconds.

TASK 11

Create the subscription:

[16]:
```
config = """{
  "Cisco-IOS-XR-telemetry-model-driven-cfg:telemetry-model-driven":  {
     "sensor-groups": {
       "sensor-group": [{
          "sensor-group-identifier": "S1",
            "sensor-paths": [{
               "sensor-path": {
                   "telemetry-sensor-path":"Cisco-IOS-XR-infra-statsd-oper:
 ↪infra-statistics/interfaces/interface/generic-counters"
               }
            }]
        }]
     },
    "destination-groups": {
        "destination-group": [{
            "destination-id": "D1",
            "ipv4-destinations": {
                "ipv4-destination": [{
                    "protocol": {
                        "protocol": "grpc",
                        "no-tls": [null]
                     },
                    "encoding": "self-describing-gpb",
                    "ipv4-address": "10.58.50.220",
                    "destination-port": 57001
                }]
            }
        }]
    },
    "subscriptions": {
        "subscription": [{
            "sensor-profiles": {
                "sensor-profile": [{
                    "sensorgroupid": "S1",
                    "sample-interval": 10000
                }]
            },
            "destination-profiles": {
                "destination-profile": [{
                    "destination-id": "D1"
                }]
            },
            "subscription-identifier": "Sub1"
        }]
    }
  }
}"""

client.set_json(config)
```

```
DEBUG:cisco_gnmi.xr:Handling update_json_configs as JSON string.
DEBUG:cisco_gnmi.client:update {
  path {
    origin: "Cisco-IOS-XR-telemetry-model-driven-cfg"
    elem {
      name: "telemetry-model-driven"
    }
  }
  val {
    json_ietf_val: "{\"sensor-groups\": {\"sensor-group\": [{\"sensor-group-
identifier\": \"S1\", \"sensor-paths\": [{\"sensor-path\": {\"telemetry-sensor-
path\": \"Cisco-IOS-XR-infra-statsd-oper:infra-
statistics/interfaces/interface/generic-counters\"}}]}]}, \"destination-
groups\": {\"destination-group\": [{\"destination-id\": \"D1\",
\"ipv4-destinations\": {\"ipv4-destination\": [{\"protocol\": {\"protocol\":
\"grpc\", \"no-tls\": [null]}, \"encoding\": \"self-describing-gpb\",
\"ipv4-address\": \"10.58.50.220\", \"destination-port\": 57001}]}}]},
\"subscriptions\": {\"subscription\": [{\"sensor-profiles\": {\"sensor-
profile\": [{\"sensorgroupid\": \"S1\", \"sample-interval\": 10000}]},
\"destination-profiles\": {\"destination-profile\": [{\"destination-id\":
\"D1\"}]}, \"subscription-identifier\": \"Sub1\"}]}}"
  }
}
```

[16]:
```
response {
  path {
    origin: "Cisco-IOS-XR-telemetry-model-driven-cfg"
    elem {
      name: "telemetry-model-driven"
    }
  }
  message {
  }
  op: UPDATE
}
message {
}
timestamp: 1624086317426458450
```

TASK 12

Confirm that the new MDT **configuration** has been saved on the device:

[17]:
```
xpath = 'Cisco-IOS-XR-telemetry-model-driven-cfg:telemetry-model-driven'
get_xpath(client, xpath)
```

```
DEBUG:cisco_gnmi.client:path {
  origin: "Cisco-IOS-XR-telemetry-model-driven-cfg"
  elem {
    name: "telemetry-model-driven"
  }
}
encoding: JSON_IETF

timestamp: 1624086323824430164
path: telemetry-model-driven
{
    "destination-groups": {
        "destination-group": [
            {
                "destination-id": "D1",
                "ipv4-destinations": {
                    "ipv4-destination": [
                        {
                            "destination-port": 57001,
                            "encoding": "self-describing-gpb",
                            "ipv4-address": "10.58.50.220",
                            "protocol": {
                                "no-tls": [
                                    null
                                ],
                                "protocol": "grpc"
                            }
                        }
                    ]
                }
            }
        ]
    },
    "sensor-groups": {
        "sensor-group": [
            {
                "sensor-group-identifier": "S1",
                "sensor-paths": {
                    "sensor-path": [
                        {
                            "telemetry-sensor-path": "Cisco-IOS-XR-infra-statsd-
oper:infra-statistics/interfaces/interface/generic-counters"
                        }
                    ]
                }
            }
        ]
    },
    "subscriptions": {
        "subscription": [
            {
                "destination-profiles": {
                    "destination-profile": [
                        {
                            "destination-id": "D1"
                        }
                    ]
                },
                "sensor-profiles": {
                    "sensor-profile": [
                        {
                            "sample-interval": 10000,
                            "sensorgroupid": "S1"
                        }
                    ]
                },
                "subscription-identifier": "Sub1"
            }
        ]
    }
}
```

Retrieve the **operational state** of the telemetry streaming subscription. Notice that the subscription is *active*:

[18]:
```
xpath = 'Cisco-IOS-XR-telemetry-model-driven-oper:telemetry-model-driven'
get_xpath(client, xpath)
```

```
DEBUG:cisco_gnmi.client:path {
  origin: "Cisco-IOS-XR-telemetry-model-driven-oper"
  elem {
    name: "telemetry-model-driven"
  }
}
encoding: JSON_IETF

timestamp: 1624086334670286070
path: telemetry-model-driven
{
    "channel-statistics": {
        "channel-statistic": [
            {
                "channel-id": "7",
                "destination-address": "10.58.50.220",
                "destination-port": 57001,
                "dropped-messages": 0,
                "encoding": "self-describing-gpb",
                "in-use-buffers": 0,
                "state": "dest-active",
                "subscription-id": "Sub1",
                "transport": "grpc"
            }
        ]
    },
    "destinations": {
        "destination": [
            {
                "configured": 1,
                "destination": [
                    {
                        "collection-group": [
                            {
                                "avg-total-time": 204,
                                "cadence": 10000,
                                "collection-path": [
                                    {
                                        "path": "Cisco-IOS-XR-infra-statsd-
oper:infra-statistics/interfaces/interface/generic-counters",
                                        "state": true
                                    }
                                ],
                                "encoding": "self-describing-gpb",
                                "id": "11",
                                "internal-collection-group": [
                                    {
                                        "avg-collection-time": "176",
                                        "cadence": "10000",
                                        "collection-method": "1",
                                        "max-collection-time": "200",
                                        "min-collection-time": "152",
                                        "path": "/oper/stats/ifg/*/generic",
                                        "status": "active",
                                        "total-collections": "2",
                                        "total-collections-missed": "0",
                                        "total-datalist-count": "0",
                                        "total-datalist-errors": "0",
                                        "total-encode-errors": "0",
                                        "total-encode-notready": "0",
                                        "total-finddata-count": "0",
                                        "total-finddata-errors": "0",
                                        "total-get-bulk-count": "0",
                                        "total-get-bulk-errors": "0",
                                        "total-get-count": "16",
                                        "total-get-errors": "0",
                                        "total-item-count": "16",
                                        "total-list-count": "2",
                                        "total-list-errors": "0",
                                        "total-send-bytes-dropped": "0",
                                        "total-send-drops": "0",
                                        "total-send-errors": "0",
                                        "total-send-packets": "2",
                                        "total-sent-bytes": "15890"
                                    }
                                ],
                                "last-collection-end-time":
"1624086326817516115",
                                "last-collection-start-time":
"1624086326640150193",
                                "max-collection-time": 200,
                                "max-total-time": 231,
                                "min-collection-time": 152,
                                "min-total-time": 177,
                                "strict-timer": false,
                                "total-collections": 2,
                                "total-not-ready": 0,
                                "total-on-data-instances": 2,
                                "total-other-errors": 0,
                                "total-send-drops": 0,
                                "total-send-errors": 0
                            }
                        ],
                        "destination": [
                            {
                                "cadence-tokens": 748,
                                "collection-tokens": 748,
                                "dest-ip-address": {
                                    "ip-type": "ipv4",
                                    "ipv4-address": "10.58.50.220"
                                },
                                "dest-port": 57001,
                                "dscp": 0,
                                "encoding": "self-describing-gpb",
                                "event-tokens": 750,
                                "id": "2009",
                                "last-collection-time": "1624086326817222434",
                                "maximum-tokens": 4000,
                                "pending-events": 0,
                                "pending-queue-size": 0,
                                "processed-events": "0",
                                "state": "dest-active",
                                "sub-id-str": "Sub1",
                                "tls": 0,
                                "total-num-of-bytes-sent": "15890",
                                "total-num-of-packets-sent": "2",
                                "transport": "grpc",
                                "udp-mtu": 0,
                                "vrf-id": 0
                            }
                        ]
                    }
                ],
                "destination-id": "D1",
                "id": "D1"
            }
```

```
        ]
    },
    "sensor-groups": {
        "sensor-group": [
            {
                "configured": 1,
                "id": "S1",
                "sensor-group-id": "S1",
                "sensor-path": [
                    {
                        "path": "Cisco-IOS-XR-infra-statsd-oper:infra-
statistics/interfaces/interface/generic-counters",
                        "state": true
                    }
                ]
            }
        ]
    },
    "subscriptions": {
        "subscription": [
            {
                "collection-group": [
                    {
                        "avg-total-time": 204,
                        "cadence": 10000,
                        "collection-path": [
                            {
                                "path": "Cisco-IOS-XR-infra-statsd-oper:infra-
statistics/interfaces/interface/generic-counters",
                                "state": true
                            }
                        ],
                        "encoding": "self-describing-gpb",
                        "id": "11",
                        "internal-collection-group": [
                            {
                                "avg-collection-time": "176",
                                "cadence": "10000",
                                "collection-method": "1",
                                "max-collection-time": "200",
                                "min-collection-time": "152",
                                "path": "/oper/stats/ifg/*/generic",
                                "status": "active",
                                "total-collections": "2",
                                "total-collections-missed": "0",
                                "total-datalist-count": "0",
                                "total-datalist-errors": "0",
                                "total-encode-errors": "0",
                "total-encode-notready": "0",
                "total-finddata-count": "0",
                "total-finddata-errors": "0",
                "total-get-bulk-count": "0",
                "total-get-bulk-errors": "0",
                "total-get-count": "16",
                "total-get-errors": "0",
                "total-item-count": "16",
                "total-list-count": "2",
                "total-list-errors": "0",
                "total-send-bytes-dropped": "0",
                "total-send-drops": "0",
                "total-send-errors": "0",
                "total-send-packets": "2",
                "total-sent-bytes": "15890"
            }
        ],
        "last-collection-end-time": "1624086326817516115",
        "last-collection-start-time": "1624086326640150193",
        "max-collection-time": 200,
        "max-total-time": 231,
        "min-collection-time": 152,
        "min-total-time": 177,
        "strict-timer": false,
        "total-collections": 2,
        "total-not-ready": 0,
        "total-on-data-instances": 2,
        "total-other-errors": 0,
        "total-send-drops": 0,
        "total-send-errors": 0
    }
],
"subscription": {
    "destination-grp": [
        {
            "configured": 1,
            "destination": [
                {
                    "cadence-tokens": 748,
                    "collection-tokens": 748,
                    "dest-ip-address": {
                        "ip-type": "ipv4",
                        "ipv4-address": "10.58.50.220"
                    },
                    "dest-port": 57001,
                    "dscp": 0,
                    "encoding": "self-describing-gpb",
                    "event-tokens": 750,
                                    "id": "2009",
                                    "last-collection-time":
"1624086326817222434",
                                    "maximum-tokens": 4000,
                                    "pending-events": 0,
                                    "pending-queue-size": 0,
                                    "processed-events": "0",
                                    "state": "dest-active",
                                    "sub-id-str": "Sub1",
                                    "tls": 0,
                                    "total-num-of-bytes-sent": "15890",
                                    "total-num-of-packets-sent": "2",
                                    "transport": "grpc",
                                    "udp-mtu": 0,
                                    "vrf-id": 0
                                }
                            ],
                            "id": "D1"
                        }
                    ],
                    "id": "Sub1",
                    "sensor-profile": [
                        {
                            "heartbeat-interval": 0,
                            "sample-interval": 10000,
                            "sensor-group": {
                                "configured": 1,
                                "id": "S1",
                                "sensor-path": [
                                    {
                                        "path": "Cisco-IOS-XR-infra-statsd-
oper:infra-statistics/interfaces/interface/generic-counters",
                                        "state": true
                                    }
                                ]
                            },
                            "suppress-redundant": false
                        }
                    ],
                    "source-interface": {
                        "ipv4-address": "0.0.0.0",
                        "ipv6-address": "::",
                        "state": false,
                        "vrf-id": 0
                    },
                    "source-qos-marking": "dscp-default",
                    "state": "active"
                },
                "subscription-id": "Sub1"
            }
        ]
    },
    "summary": {
        "max-containers-per-path": 16,
        "max-sensor-paths": 1000,
        "min-target-def-cadence": 30000,
        "num-of-active-destinations": 1,
        "num-of-active-sensor-paths": 1,
        "num-of-active-subscriptions": 1,
        "num-of-connected-sessions": 1,
        "num-of-connecting-sessions": 0,
        "num-of-destination-groups": 1,
        "num-of-destinations": 1,
        "num-of-dialins": 0,
        "num-of-grpc-non-tls-dialouts": 1,
        "num-of-grpc-tls-dialouts": 0,
        "num-of-not-resolved-sensor-paths": 0,
        "num-of-paused-subscriptions": 0,
        "num-of-sensor-groups": 1,
        "num-of-sensor-paths": 1,
        "num-of-subscriptions": 1,
        "num-of-tcp-dialouts": 0,
        "num-of-udp-dialouts": 0,
        "num-of-unique-sensor-paths": 1,
        "target-def-cadence-factor": 2
    }
}
```

The host 198.18.134.50 already has the collection components for ingesting, storing and visualizing telemetry data. We are using the following set of Docker containers: - Telegraf - a collection container that runs the cisco_telemetry_mdt plugin for ingestion of model-driven telemetry data - InfluxDB - a storage container that runs a database which stores the data - Chronograf - a visualization container that runs a web application which allows the exploration of data

The collector listens on port 57000 for incoming connections. IOS XR will initiate the connection, dialling out to the collector. After the connection establishment, the device will begin to stream data over gRPC to the collector which will send it to a database named telemetry.

NOTE - Please refer to the following references for details about the three components and the ingestion of telemetry data: - InfluxData - Input plugin for Model-driven telemetry over gRPC - Input plugin for Model-driven telemetry over gNMI

TASK 13

In a separate browser tab, open the web application for data visualization located at http://198.18.134.50:8888. Then, hover over the left hand-side and click on Dashboards. Open the `Traffic Monitoring` dashboard.

NOTE - If you are missing the dashboard, use the `Import Dashboard` button to upload the dashboard located at `monitoring/chronograf/Traffic Monitoring.json`.

TASK 14

In the next task, we will learn about MDT streaming triggered with the dial-in mechanism. Remove the current dial-out configuration:

```
[19]: xpath = 'Cisco-IOS-XR-telemetry-model-driven-cfg:telemetry-model-driven'
      client.delete_xpaths(xpath)
```

```
DEBUG:cisco_gnmi.client:delete {
  origin: "Cisco-IOS-XR-telemetry-model-driven-cfg"
  elem {
    name: "telemetry-model-driven"
  }
}
```

```
[19]: response {
        path {
          origin: "Cisco-IOS-XR-telemetry-model-driven-cfg"
          elem {
            name: "telemetry-model-driven"
          }
        }
        message {
        }
        op: DELETE
      }
      message {
      }
      timestamp: 1624086397399809016
```

TASK 15

Confirm that the configuration has been deleted:

```
[20]: xpath = 'Cisco-IOS-XR-telemetry-model-driven-cfg:telemetry-model-driven'
      get_xpath(client, xpath)
```

```
DEBUG:cisco_gnmi.client:path {
  origin: "Cisco-IOS-XR-telemetry-model-driven-cfg"
  elem {
    name: "telemetry-model-driven"
  }
}
encoding: JSON_IETF

timestamp: 1624086401526851626
path: telemetry-model-driven
```

0.1.7 Configuration of model-driven telemetry streaming (dynamic, dial-in)

TASK 16

Open the `cisco-gNMI-dynamic-subscription` Jupyter notebook and run it to create a subscription that will be active throughout the laboratory.

TASK 23

Check if there is model-driven telemetry **configuration** on the device:

```
[21]: xpath = 'Cisco-IOS-XR-telemetry-model-driven-cfg:telemetry-model-driven'
      get_xpath(client, xpath)
```

```
DEBUG:cisco_gnmi.client:path {
  origin: "Cisco-IOS-XR-telemetry-model-driven-cfg"
  elem {
    name: "telemetry-model-driven"
  }
}
encoding: JSON_IETF

timestamp: 1624086483169832150
path: telemetry-model-driven
```

Notice that there is no actual telemetry configured on the device. The gNMI-based subscription was created dynamically by the gNMI client, without touching the configuration of the device. Instead, this host has instantiated a **dynamic** subscription over the RPC channel. This subscription does not have local persistence on the IOS XR device, and is not persistent between device reboots. However, the existence of the dynamic subscription is marked in the operational state of the device.

NOTE - For more information about persistent and dynamic subscription, see the `openconfig-telemetry.yang` YANG model that the gNMI implementation used as reference for the subscription functionality: - persistent: https://github.com/openconfig/public/blob/master/release/models/telemetry/openconfig-telemetry.yang#L247 - dynamic: https://github.com/openconfig/public/blob/master/release/models/telemetry/o
telemetry.yang#L414

Let's retrieve the **operational state** of telemetry streaming with the following command:

```
[22]: xpath = "Cisco-IOS-XR-telemetry-model-driven-oper:telemetry-model-driven"
      get_xpath(client, xpath)
```

```
DEBUG:cisco_gnmi.client:path {
  origin: "Cisco-IOS-XR-telemetry-model-driven-oper"
  elem {
    name: "telemetry-model-driven"
  }
}
encoding: JSON_IETF

timestamp: 1624086487494736168
```

```
path: telemetry-model-driven
{
    "channel-statistics": {
        "channel-statistic": [
            {
                "channel-id": "2",
                "destination-address": "10.61.66.183",
                "destination-port": 64789,
                "dropped-messages": 0,
                "encoding": "gnmi-proto",
                "in-use-buffers": 0,
                "state": "dest-active",
                "subscription-id": "GNMI__8892453317995001798",
                "transport": "dialin"
            }
        ]
    },
    "destinations": {
        "destination": [
            {
                "configured": 0,
                "destination": [
                    {
                        "collection-group": [
                            {
                                "avg-total-time": 16,
                                "cadence": 60000,
                                "collection-path": [
                                    {
                                        "path": "Cisco-IOS-XR-infra-statsd-
oper:infra-statistics/interfaces/interface/generic-counters",
                                        "state": true
                                    }
                                ],
                                "encoding": "gnmi-proto",
                                "id": "12",
                                "internal-collection-group": [
                                    {
                                        "avg-collection-time": "16",
                                        "cadence": "60000",
                                        "collection-method": "1",
                                        "max-collection-time": "16",
                                        "min-collection-time": "16",
                                        "path": "/oper/stats/ifg/*/generic",
                                        "status": "active",
                                        "total-collections": "1",
                                        "total-collections-missed": "0",
                                        "total-datalist-count": "0",
                                "total-datalist-errors": "0",
                                "total-encode-errors": "0",
                                "total-encode-notready": "0",
                                "total-finddata-count": "0",
                                "total-finddata-errors": "0",
                                "total-get-bulk-count": "0",
                                "total-get-bulk-errors": "0",
                                "total-get-count": "1",
                                "total-get-errors": "0",
                                "total-item-count": "1",
                                "total-list-count": "1",
                                "total-list-errors": "0",
                                "total-send-bytes-dropped": "0",
                                "total-send-drops": "0",
                                "total-send-errors": "0",
                                "total-send-packets": "1",
                                "total-sent-bytes": "1276"
                            }
                        ],
                        "last-collection-end-time":
"1624086470569952848",
                        "last-collection-start-time":
"1624086470553035023",
                        "max-collection-time": 16,
                        "max-total-time": 16,
                        "min-collection-time": 16,
                        "min-total-time": 16,
                        "strict-timer": false,
                        "total-collections": 1,
                        "total-not-ready": 0,
                        "total-on-data-instances": 0,
                        "total-other-errors": 0,
                        "total-send-drops": 0,
                        "total-send-errors": 0
                    }
                ],
                "destination": [
                    {
                        "cadence-tokens": 748,
                        "collection-tokens": 749,
                        "dest-ip-address": {
                            "ip-type": "ipv4",
                            "ipv4-address": "10.61.66.183"
                        },
                        "dest-port": 64789,
                        "dscp": 0,
                        "encoding": "gnmi-proto",
                        "event-tokens": 750,
                                "gnmi-sync-response-time": "2021-06-19
07:07:52.570376 +0000",
                                "id": "2010",
                                "last-collection-time": "1624086472570306720",
                                "maximum-tokens": 4000,
                                "pending-events": 0,
                                "pending-queue-size": 0,
                                "processed-events": "0",
                                "state": "dest-active",
                                "sub-id-str": "GNMI__8892453317995001798",
                                "tls": 1,
                                "total-num-of-bytes-sent": "1276",
                                "total-num-of-packets-sent": "2",
                                "transport": "dialin",
                                "udp-mtu": 0,
                                "vrf-id": 0
                            }
                        ]
                    }
                ],
                "destination-id": "GNMI_1006",
                "id": "GNMI_1006"
            }
        ]
    },
    "sensor-groups": {
        "sensor-group": [
            {
                "configured": 0,
                "id": "GNMI__8892453317995001798_0",
                "sensor-group-id": "GNMI__8892453317995001798_0",
                "sensor-path": [
                    {
                        "path": "Cisco-IOS-XR-infra-statsd-oper:infra-
statistics/interfaces/interface[interface-name=MgmtEth0/RP0/CPU0/0]/generic-
counters",
                        "state": true
                    }
                ]
            }
        ]
    },
    "subscriptions": {
        "subscription": [
            {
                "collection-group": [
                    {
                        "avg-total-time": 16,
                        "cadence": 60000,
                        "collection-path": [
                            {
                                "path": "Cisco-IOS-XR-infra-statsd-oper:infra-
statistics/interfaces/interface/generic-counters",
                                "state": true
                            }
                        ],
                        "encoding": "gnmi-proto",
                        "id": "12",
                        "internal-collection-group": [
                            {
                                "avg-collection-time": "16",
                                "cadence": "60000",
                                "collection-method": "1",
                                "max-collection-time": "16",
                                "min-collection-time": "16",
                                "path": "/oper/stats/ifg/*/generic",
                                "status": "active",
                                "total-collections": "1",
                                "total-collections-missed": "0",
                                "total-datalist-count": "0",
                                "total-datalist-errors": "0",
                                "total-encode-errors": "0",
                                "total-encode-notready": "0",
                                "total-finddata-count": "0",
                                "total-finddata-errors": "0",
                                "total-get-bulk-count": "0",
                                "total-get-bulk-errors": "0",
                                "total-get-count": "1",
                                "total-get-errors": "0",
                                "total-item-count": "1",
                                "total-list-count": "1",
                                "total-list-errors": "0",
                                "total-send-bytes-dropped": "0",
                                "total-send-drops": "0",
                                "total-send-errors": "0",
                                "total-send-packets": "1",
                                "total-sent-bytes": "1276"
                            }
                        ],
                        "last-collection-end-time": "1624086470569952848",
                        "last-collection-start-time": "1624086470553035023",
                        "max-collection-time": 16,
                        "max-total-time": 16,
                        "min-collection-time": 16,
                        "min-total-time": 16,
                        "strict-timer": false,
                        "total-collections": 1,
                        "total-not-ready": 0,
                        "total-on-data-instances": 0,
                        "total-other-errors": 0,
                        "total-send-drops": 0,
                        "total-send-errors": 0
                    }
                ],
                "subscription": {
                    "destination-grp": [
                        {
                            "configured": 0,
                            "destination": [
                                {
                                    "cadence-tokens": 748,
                                    "collection-tokens": 749,
                                    "dest-ip-address": {
                                        "ip-type": "ipv4",
                                        "ipv4-address": "10.61.66.183"
                                    },
                                    "dest-port": 64789,
                                    "dscp": 0,
                                    "encoding": "gnmi-proto",
                                    "event-tokens": 750,
                                    "gnmi-sync-response-time": "2021-06-19
07:07:52.570376 +0000",
                                    "id": "2010",
                                    "last-collection-time":
"1624086472570306720",
                                    "maximum-tokens": 4000,
                                    "pending-events": 0,
                                    "pending-queue-size": 0,
                                    "processed-events": "0",
                                    "state": "dest-active",
                                    "sub-id-str": "GNMI__889245331",
                                    "tls": 1,
                                    "total-num-of-bytes-sent": "1276",
                                    "total-num-of-packets-sent": "2",
                                    "transport": "dialin",
                                    "udp-mtu": 0,
                                    "vrf-id": 0
                                }
                            ],
                            "id": "GNMI_1006"
                        }
                    ],
                    "id": "GNMI__8892453317995001798",
                    "sensor-profile": [
                        {
                            "heartbeat-interval": 0,
                            "sample-interval": 60000,
                            "sensor-group": {
                                "configured": 0,
                                "id": "GNMI__8892453317995001798_0",
                                "sensor-path": [
                                    {
                                        "path": "Cisco-IOS-XR-infra-statsd-
oper:infra-statistics/interfaces/interface[interface-
name=MgmtEth0/RP0/CPU0/0]/generic-counters",
                                        "state": true
                                    }
                                ]
                            },
                            "suppress-redundant": false
                        }
                    ],
                    "source-interface": {
                        "ipv4-address": "0.0.0.0",
                        "ipv6-address": "::",
                        "state": false,
                        "vrf-id": 0
                    },
                    "source-qos-marking": "dscp-default",
                    "state": "active"
                },
                "subscription-id": "GNMI__8892453317995001798"
            }
        ]
    },
    "summary": {
        "max-containers-per-path": 16,
        "max-sensor-paths": 1000,
        "min-target-def-cadence": 30000,
        "num-of-active-destinations": 1,
        "num-of-active-sensor-paths": 1,
        "num-of-active-subscriptions": 1,
        "num-of-connected-sessions": 1,
        "num-of-connecting-sessions": 0,
        "num-of-destination-groups": 1,
        "num-of-destinations": 1,
        "num-of-dialins": 1,
        "num-of-grpc-non-tls-dialouts": 0,
        "num-of-grpc-tls-dialouts": 1,
        "num-of-not-resolved-sensor-paths": 0,
        "num-of-paused-subscriptions": 0,
        "num-of-sensor-groups": 1,
        "num-of-sensor-paths": 1,
        "num-of-subscriptions": 1,
        "num-of-tcp-dialouts": 0,
        "num-of-udp-dialouts": 0,
        "num-of-unique-sensor-paths": 1,
        "target-def-cadence-factor": 2
    }
}
```

Notice that there is a new *active* subscription reported. The values of the IDs for the sensor-group, destination-group and subscription request that were created as a result of the dynamic subscription request follow the format: **GNMI___XYZ**.