Network Automation Made Easy

ciscopress.com

IVO PINTO, CCIE® NO. 57162

Network Automation Made Easy

Ivo Pinto, CCIE No. 57162

Cisco Press

```
            <ipv6 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip" />
         </interface>
      </interfaces>
   </data>
</rpc-reply>
```

You mostly need to be familiar with what is inside the operation tags, as the outer parts of the payload *<rpc>* and operation tags are typically abstracted by the tool you use to interact with a NETCONF-enabled device (unless you are using SSH directly). Say that you want to configure an interface description and an IP address on the interface Gigabit Ethernet 1/1 of an IOS XE device. You could achieve this by using the payload shown in Example 2-28.

Example 2-28 *NETCONF Code to Configure an Interface*

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="">
   <edit-config>
      <target>
         <running />
      </target>
      <config>
         <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
            <interface>
               <name>GigabitEthernet1/1</name>
               <description>new description</description>
               <ipv4 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip">
                  <address>
                     <ip>10.0.0.1</ip>
                     <netmask>255.255.255.0</netmask>
                  </address>
               </ipv4>
            </interface>
         </interfaces>
      </config>
   </edit-config>
</rpc>
```

When using the ncclient Python module, you pass the payload shown in Example 2-29.

Example 2-29 *Python Payload to Configure an Interface Using NETCONF*

```
<config>
         <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
            <interface>
               <name>GigabitEthernet1/1</name>
               <description>new description</description>
               <ipv4 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip">
                  <address>
                     <ip>10.0.0.1</ip>
                     <netmask>255.255.255.0</netmask>
                  </address>
               </ipv4>
            </interface>
         </interfaces>
</config>
```

In Example 2-29, you can see that all the outer tags are gone. You inform the tool about the desired operation and datastore in the command itself. *edit_config* is the operation, and the *target* parameter represents the target datastore:

```
edit_config(xml_payload, target="running"
```

As you can see in the previous examples, the resulting data is automation friendly, and it is easy to parse and consume. However, NETCONF requires quite a change from the ways people are accustomed to doing things, and it was not well adopted at first. A further evolution was RESTCONF.

RESTCONF

RESTCONF brought a REST API interface to NETCONF and YANG. It is stateless, uses HTTP as the transport method, and uses the HTTP methods described in Table 2-7. APIs, as previously mentioned, are widely adopted in the software world, and RESTCONF allows users who are familiar with APIs to use them to easily interact with network devices. The tools you use to interact with RESTCONF are the same as the API tools: **curl**, Postman, Python, and so on.

Unlike NETCONF data, RESTCONF data can be encoded using XML or JSON. The URI is an important element of REST APIs, and a RESTCONF URI is similar. It has the following format:

```
https://ADDRESS/ROOT/RESOURCE/[YANGMODULE:]CONTAINER/LEAF[?OPTIONS]
```

Table 2-7 *NETCONF URI Elements*

Datastore	Description
ADDRESS	Specifies the IP address of the RESTCONF-capable target agent.
ROOT	Specifies the main entry point for RESTCONF requests. To discover it you can use **https://***device_IP***/***.well-known***/***host-meta*.
RESOURCE	Typically specifies the value /data or /operations, representing the type of resource being accessed.
YANGMODULE and CONTAINER	Specifies the base model container being used.
LEAF	Specifies an individual element of a YANG container.
OPTIONS	(Optional) Specifies the optional parameters, such as type of content that impacts return results.

Let us revisit the NETCONF example of a Cisco IOS XE device. The URI path to retrieve its configuration using the native YANG model is as follows:

`https://device_IP/restconf/data/Cisco-IOS-XE-native:native/`

To retrieve only the interface Gigabit 1/1/13, as shown earlier in Example 2-26 using NETCONF, you make a GET request to the URI in Example 2-30. (Note, in this example, that *%2F* is the URL encoding of the symbol /, as you cannot pass this symbol natively in a URI.)

Example 2-30 *Retrieving Interface Information by Using RESTCONF*

```
curl -u 'cisco:cisco123' --request GET 'https://{{device_IP}}/restconf/data/
  ietf-interfaces:interfaces/interface=GigabitEthernet1%2F0%2F13'


<interface xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces"
  xmlns:if="urn:ietf:params:xml:ns:yang:ietf-interfaces">
  <name>GigabitEthernet1/0/13</name>
  <description>Fabric Physical Link</description>
  <type xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-
  type">ianaift:ethernetCsmacd</type>
  <enabled>true</enabled>
  <ipv4 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip">
    <address>
      <ip>172.31.63.165</ip>
      <netmask>255.255.255.254</netmask>
    </address>
  </ipv4>
  <ipv6 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip">
  </ipv6>
</interface>
```

You may notice that the result is exactly the same result obtained previously using NETCONF, as both examples use the same device. The structure represented on the URI maps directly to the YANG module used. In this example, you are using the ietf-interfaces YANG module and the interfaces container within in. The leaf is the interface named Gigabit Ethernet 1/1/13. You can see the importance of understanding a little bit of YANG.

To make configuration changes, you can use PUT, POST, or PATCH, as discussed earlier in this chapter. To replace the current interface configuration with a different one, for example, we could execute the PUT operation by using **curl**, as shown in Example 2-31. Notice that the changes go in the payload, and the URI is maintained.

Example 2-31 *Modifying Interface Configurations by Using RESTCONF*

```
curl --request PUT 'https://{{device_IP}}/restconf/data/ietf-interfaces:interfaces/
  interface=GigabitEthernet1%2F0%2F13' \
--header 'Content-Type: application/yang-data+xml' \
-u 'cisco:cisco123' \
--data-raw '<interface xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces"
  xmlns:if="urn:ietf:params:xml:ns:yang:ietf-interfaces">
    <name>GigabitEthernet1/0/13</name>
    <description>Fabric Physical Link</description>
    <type xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-
  type">ianaift:ethernetCsmacd</type>
    <enabled>true</enabled>
    <ipv4 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip">
        <address>
            <ip>172.31.63.164</ip>
            <netmask>255.255.255.254</netmask>
        </address>
    </ipv4>
    <ipv6 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip">
  </ipv6>
</interface>'
```

In this example, you receive an empty 204 response. Notice here that you have sent all the information for PUT, even the information you did not want to change; this is a replace operation, as explained earlier in this chapter, and so any omitted data will be replaced by its default value.

If you simply want to change the description of an interface, you can use PATCH and a simpler payload:

```
<interface xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces"
xmlns:if="urn:ietf:params:xml:ns:yang:ietf-interfaces">
    <description>New description</description>
</interface>
```

By now you have a good understanding of NETCONF and RESTCONF. Figure 2-6 shows a summary of these two protocols, as well as the Google protocol Google RPC (gRPC), which you will see later in this chapter.

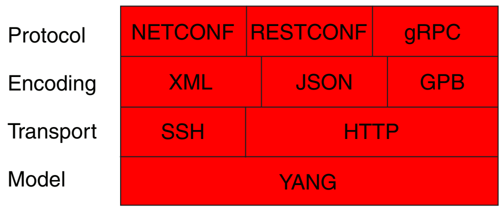

Figure 2-6 *Model-Driven Protocol Stack*

Telemetry

This chapter has covered model-driven techniques from a data configuration standpoint. One area where these techniques are commonly used is with telemetry. Telemetry is not configuration data but metric data. In the past, SNMP was the de facto standard for retrieving this type of data, and CLI connection methods such as SSH were also used. However, some use cases, such as setting off alarms based on critical conditions, require instant action on data, and most methods cannot achieve such agility. Model-driven telemetry is a push-based mechanism (refer to Chapter 1) that addresses the shortcomings of polling mechanisms such as SNMP.

There are two categories of model-driven telemetry:

- **Dial-in:** Telemetry management systems subscribe to telemetry streams on the devices.
- **Dial-out:** A device configures the subscription.

In either category, telemetry-enabled devices constantly send telemetry data to their management systems, which store this data in database systems. This enables access to real-time, model-driven data, which can then be used in network automation systems.

There are both periodic and change-based notifications. That is, a management system can receive information constantly every specified amount of time, or it can be notified only when a change occurs in a specific model.

The most commonly used data model for network telemetry is YANG. Hence, the protocols used are NECONF, RESTCONF, and gRPC.

Telemetry has not yet been widely adopted, but it is quickly taking off as vendors enable their devices for this type of access.

One way of configuring model-driven telemetry is directly on a device (dial-out). Consider the following example on a Cisco CSR 1000V:

```
csrv000v(config)#telemetry ietf subscription 102
csr1000v(config-mdt-subs)#encoding encode-kvgpb
```

```
csr1000v(config-mdt-subs)#filter xpath /process-cpu-ios-xe-oper:cpu-
usage/cpu-utilization/five-seconds
csr1000v(config-mdt-subs)#stream yang-push
csr1000v(config-mdt-subs)#update-policy periodic 1000
csr1000v(config-mdt-subs)#receiver ip address 10.0.0.1 57500 protocol
grpc-tcp
```

This example shows data being sent to a collector at 10.0.0.1, using gRPC and Google protocol buffers as encoding. Data on CPU utilization is being sent every 10 seconds (using the process-cpu-ios-xe-oper YANG module).

You can also configure telemetry subscriptions by using NETCONF or RESTCONF. At the time of this writing, only gRPC is supported for dial-out telemetry in IOS XE.

The collector must be configured to receive this type of data. Commonly used collectors are Telegraf and Splunk. After the data is received in the tool, it should be stored. Finally, from the stored data, you can create visualizations and alarms (for example, by using Grafana). You will see more about this topic in Chapter 3, "Using Data from Your Network."

Tip Verify whether your devices support model-driven telemetry. If they do, use model-driven telemetry instead of SNMP or the CLI.

Log Exporters

This section covers exporters for two types of data:

- Message logs
- Flow logs

Message logs are vast in networks. They can originate in networking equipment, servers, applications, and other places. Message data type is different from metrics in that it is more verbose and typically bigger in size. Independently of the format or type, logs are good indicators of system behaviors. Most systems are programmed to issue logs when events such as failures or alarming conditions occur.

In networking, the most common log format is Syslog, covered earlier in this chapter.

The way to set up log exports differs from system to system. However, most networking systems today support a Syslog destination server. A typical configuration on an end system, such as a Cisco router, would look as follows, with a destination server IP address:

```
Router(config)# logging host 10.0.0.1
```

The destination IP address should point at a central collection tool, such as one of the tools mentioned in Chapter 1. The Syslog server can then process these logs and either display them all together in a dashboard or redirect the relevant ones to other tools; for example, it might send the security logs to the security team monitoring tool and the operational logs to a different tool.