NETCONF

Network Configuration Protocol

one of the southbond APIs between OpenDayLight and network devices

This document covers the following contents:

Overview on NETCONF, XML, and YANG. Motivation to know about NETCONF. Some understanding on Layers, Session, Operations, Parameters and Attributes, Standard configuration procedure. Sample comware configuration snippets / python codes that can help with a Practical illustration

NETCONF Overview

- NETCONF is an XML RPC (remote procedure call) based protocol
- RPC is a protocol that one program (on the client) can use to request a service from a program located in another computer (server / networking device) on a network without having to understand the network's details
- The NETCONF protocol uses an XML-based data encoding for the configuration data and remote procedure calls
- NETCONF was first brought to the IETF by Juniper with their XML configuration management concept and shared with the community

XML Overview

- The typical CLI format is *clear text*, which is easy for humans to read and interpret. However, it is terrible for computer programs to interpret because it lacks "structure" and often uses white space and position information to indicate meaning. It is almost impossible to make clear text data consistent across multiple platforms, particularly with multiple vendors
- XML stands for eXtensible Markup Language, it is designed to store(in plain text format) and transport data, it is both human and machine readable, and is often used for distributing data over the Internet
- Because of its highly-structured format, developers building applications or scripts to consume XML don't need to be as concerned with parsing white space and position to identify the information they are seeking
- XML is just information wrapped in tags, it does not do anything by itself. Someone must write a piece of software using a programming language like python, perl, javascript, go etc. to send, receive, store, or display XML
- The XML language has no predefined tags. With XML, the author must define both the tags and the document structure.
- XML Namespaces provide a method to avoid element name conflicts. In XML, element names are defined by the developer. This often results in a conflict when trying to mix XML documents from different XML applications.

YANG Overview

- Before YANG models existed, NETCONF used vendor-specific data models
- YANG is a data modeling language, used by NETCONF at its content layer to provide the human readable format of the content (example: Configuration Data, State Data etc.)
- when we are working with any NETCONF based system, we will have typically some no. of YANG models that forms the entire model set for that system.
- YANG is an xml schema definition language, it maps one-to-one to xml, it defines how the data is represented in XML and how it is used in NETCONF operations
- It defines a data model in terms of the data, its hierarchial organization, and the constraints on that data
- There are IETF drafts suggesting YANG data models for interface configuraton, IP configuration, etc.
- Types of YANG data models – Device (Interface, VLAN, OSPF, etc.), Service (L3 MPLS VPN, MP-BGP, etc.)

Why NETCONF

- NETCONF is seen to be more configuration specific, SNMP is likely to be used for its Notifications, Traps, etc. so NETCONF is not aiming at replacing SNMP
- NETCONF is a configuration protocol, while OpenFlow operates only on the flow tables that specify how incoming packets are to be routed. If an OpenFlow switch exposes the schema to configure the flow tables, then NETCONF can also be used to program the forwarding tables of the switch
- An XML file with standard parameters or attributes, can be used to modify the configuration of any NETCONF server enabled networking device irrespective of the vendor
- Growing networking vendor support
- more developments in the client applications/tools area, both opensource and commercial

Layers

4. content: State/Operational data, configuration data etc.

3. Operations: <get>, <get-config> etc.

2. Messages: <hello> <rpc> <rpc-reply> <notification>

1. secure transport: BEEP, SSH, SSL, console, TLS, SOAP, etc.

SSH SubSystem

- majority of the NETCONF implementations use SSH, alternatives are SOAP, TLS etc.
- Subsystem support is only a feature of SSH v2, so NETCONF can only work with SSH v2 not v1
- ssh subsystem, default TCP port 830

Session

- Once the SSH-connection service is established, the client application will open a channel of type ‘session’ resulting in a SSH session
- As soon as the SSH session is established, the server and the client will simultaneously exchange their capabilities (example: candidate configuration datastore support by the server) with the first XML document/message containing a <hello> element
- The capabilities information in the XML hello message can be parsed by the receiver to see what the sender supports
- Once the session capabilities have been exchanged, the NETCONF client will send complete XML documents containing <rpc> elements to the server, and the NETCONF server will respond with complete XML documents containing <rpc-reply> elements
- The SSH Transport layer encodes messages sent by the Messages layer, and decodes messages received on the SSH channel before passing them to the Messages layer.
- After exchanging data the client application closes the session and the connection to the NETCONF server

Hello message example

Server

```
<?xml version="1.0" encoding="UTF-8"?>
<hello xmlns="urn:ietf:params:xml:ns:NETCONF:base:1.0">
  <capabilities>
    <capability>
      urn:ietf:params:NETCONF:base:1.1
    </capability>
    <capability>
      urn:ietf:params:ns:NETCONF:capability:startup:1.0
    </capability>
  </capabilities>
  <session-id>4</session-id>
</hello>
]]>]]>
```

Client

```
<?xml version="1.0" encoding="UTF-8"?>
<hello xmlns="urn:ietf:params:xml:ns:NETCONF:base:1.0">
  <capabilities>
    <capability>
      urn:ietf:params:NETCONF:base:1.1
    </capability>
  </capabilities>
</hello>
]]>]]>
```

XML RPC - example

```
<rpc>
   <edit-config>
      <target>
         <candidate/>
      </target>
      <config>
         ...New Configuration...
      </config>
   </edit-config>
</rpc>
```

```
<rpc>
  <commit\>
</rpc>
```

Data stores

- Candidate
  not the live configuration, it could be a persistent future configuration
- running
  live configuration
- startup
  during boot up

- URL – offered by cisco IOS

Operations

<get> - an operation that returns both configurational and operational data in a datastore, similar to snmp-walk

<get-config> - an operation to get only the configuration data in a datastore

<get-chassis-inventory>

<notification> - optional, to receive event notifications

<validate> - validate the content of a datastore

<edit-config> - goes directly to running config, to edit it

<copy-config> - commonly copy config is done from running to candidate

Operations contd.

<lock> and <unlock> operations - to lock and unlock a datastore, from another user

<delete-config> - clear candidate data store

discard changes and cancel commit -> only available when candidate data store is turned on

<commit> if both running and candidate are available commit or confirmed transaction could be applied to commit accumulated changes on candidate to the running datastore

<close-session>/<kill-session> gracefully/forcefully close

Parameters and Attributes

- <filter type=subtree>   filter is a parameter, type is an attribute
- two types of filters - simple subtree filter(mandatory), filter with xpath (advanced, its a whole query language). subtree doesn't suport wild cards, xpath has great features
- <target> parameter is used to specify which datastore are we manipulating
- with the <commit> operation, <confirmed> parameter, <persist> parameter is used to specify a commit identifier, <commit-timeout> parameter is used to specify a timeout before rollback

Multiple devices configuration - procedure

Step1: prepare

lock datastore(running) -> clear candidate (delete-config) -> edit candidate -> validate

Step2: commit

commit -> confirming commit -> unlock datastore

we can test the network between commit and confirming commit

close / kiliing the session can also rollback, as it also releases the lock

Comware config – NETCONF server

```
ssh server enable
NETCONF ssh server enable
NETCONF ssh server port 830
ssh user admin service-type netconf authentication-type password
```

Comware config - general

```
interface M-GigabitEthernet0/0/0
 ip address 192.168.56.11 255.255.255.0

local-user admin class manage
password simple p@ssword
service-type ssh
authorization-attribute user-role network-admin
```

```
line vty 0 63
user-role network-admin
authentication-mode scheme
protocol inbound ssh

public-key local create rsa
y
2048
```

NETCONF using linux

```
ssh username@ip -p 830 -s netconf
password:
<?xml version="1.0" encoding="UTF-8"?>
 <hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <capabilities>
   <capability>urn:ietf:params:netconf:base:1.0</capability>
 </capabilities>
   </hello>]]>]]>


 <?xml version="1.0" encoding="UTF-8"?>
 <rpc message-id="1239123"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <close-session />
 </rpc>
 ]]>]]>
```

Sample commands are shown on the left,

Once the netconf session is enabled, the xml messages, hello, rpc, etc. can be sent directly from the shell, until the session is closed using the <close-session> operation

But this way of configuring may be a little erroneous, and may require manual intervention for generating message IDs, netconf clients or libraries are considered to be better for performing configuration tasks

ncclient

- NETCONF modules are developed to be used with programming languages such as c, python, perl, go, etc.
- ncclient is one such module developed in python to use at the client side application
- ncclient offers an intuitive API that sensibly maps the XML-encoded nature of NETCONF to python constructs and idioms, and make network-management codes easier
- To install: type ‘pip install ncclient’, on the command prompt

ncclient supported device handlers

Juniper: device_params={'name':'junos'}
Cisco CSR: device_params={'name':'csr'}
Cisco Nexus: device_params={'name':'nexus'}
Cisco IOS XR: device_params={'name':'iosxr'}
Cisco IOS XE: device_params={'name':'iosxe'}
Huawei: device_params={'name':'huawei'}
Alcatel Lucent: device_params={'name':'alu'}
H3C: device_params={'name':'h3c'}
HP Comware: device_params={'name':'hpcomware'}
Server or anything not in above: device_params={'name':'default'}

Session establishment

```
from ncclient import manager
host = '192.168.56.11’
username = 'admin'
password = 'p@ssword'
c = manager.connect(host=host, port=830, username=username,
password=password, hostkey_verify=False, device_params={'name
        ':'h3c’})
```

Display server capabilities

```
>>> for capability in c.server_capabilities:
...     print capability
...
urn:ietf:params:netconf:capability:writable-running:1.0
urn:ietf:params:netconf:capability:validate:1.0
urn:ietf:params:netconf:base:1.0
urn:h3c:params:netconf:capability:h3c-netconf-ext:1.0
urn:ietf:params:netconf:capability:notification:1.0
urn:ietf:params:netconf:capability:interleave:1.0
```

Execute commands

```
display_command = '''
<Execution> display ip interface brief\ndisplay arp</Execution>
'''
c.cli(display_command)
```

Retrieve complete configuration

```
data_all = c.get()


import xml.dom.minidom
x = data_all.xml
xml_data =
xml.dom.minidom.parseString(x)
pretty_xml =
xml_data.toprettyxml()
print pretty_xml
```

```
all_config = '''
<top
xmlns="http://www.h3c.com/netc
onf/data:1.0">
'''

c.get(('subtree', all_config))
```

Retrieve running configuration

```
running_config = c.get_config(source='running')
f = open("%s.xml" % host, 'w')
f.write(running_config.data_xml)
```

Filter VLANs

```
vlan_filter = '''
<top
xmlns='http://www.h3c.com/netconf
/data:1.0'>
    <VLAN>
        <VLANs>
            </VLANs>
        </VLAN>
</top>
'''
```

```
c.get(('subtree', vlan_filter))
```

Filter VLAN IDs

```
vlanid_filter = '''
<top
xmlns='http://www.h3c.com/netconf/data:1.0’>
   <VLAN>
      <VLANs>
         <VLANID>
            <ID>
            </ID>
         </VLANID>
      </VLANs>
   </VLAN>
</top>
'''
```

```
c.get(('subtree', vlanid_filter))
```

Configure VLAN

```
add_vlan = '''
<config xmlns='urn:ietf:params:xml:ns:netconf:base:1.0'>
  <top xmlns='http://www.h3c.com/netconf/config:1.0'>
    <VLAN>
      <VLANs>
        <VLANID>
          <ID> 101 </ID>
          <Name> one-zero-one </Name>
          <Description> OnE-zErO-oNe </Description>
        </VLANID>
      </VLANs>
    </VLAN>
  </top>
</config>
'''
```

```
c.edit_config(target='running', config=add_vlan,
default_operation='replace')
```

Filter VLAN ID

```
vlanid_filter_101 = '''
<top
xmlns='http://www.h3c.com/netconf/data:1.0’>
   <VLAN>
      <VLANs>
         <VLANID>
            <ID> 101 </ID>
         </VLANID>
      </VLANs>
   </VLAN>
</top>
'''
```

```
c.get(('subtree', vlanid_filter_101))
```

Configure OSPF

```
add_OSPF = '''
<config xmlns='urn:ietf:params:xml:ns:netconf:base:1.0'>
  <top xmlns='http://www.h3c.com/netconf/config:1.0'>
    <OSPF>
      <Instances>
        <Instance>
          <Name> 100 </Name>
          <VRF/>
            <RouterId>192.168.56.103</RouterId>
        </Instance>
      </Instances>
<Areas>
        <Area>
          <Name> 100 </Name>
          <AreaId> 0.0.0.100 </AreaId>
</Area>
      </Areas>
<Networks>
        <Network>
          <Name> 100 </Name>
          <AreaId> 0.0.0.100 </AreaId>
          <NetworkAddr> 1.1.1.1 </NetworkAddr>
          <WildcardMask> 0.0.0.0 </WildcardMask>
        </Network>
      </Networks>
    </OSPF>
  </top>
</config>
'''
c.edit_config(target='running', config=add_OSPF)
```

Filter OSPF

```
ospf_filter = '''
<top
xmlns='http://www.h3c.com/netc
onf/data:1.0'>
  <OSPF>
  </OSPF>
</top>
'''
c.get(('subtree', ospf_filter))
```

```
import re
items = re.findall("OSPF.*$",
pretty_xml, re.MULTILINE)
for x in items:
    print x
```

Close the session

```
c.close_session()
```

Reference

RFC 3535

RFC 5277

RFC 6020

RFC 6241

http://www.netconfcentral.org/

https://pypi.org/project/ncclient/0.4.7/

--end-of-document--

networkandcode.wordpress.com