Software Defined Network
SDN

Ch 10

NetConf and YANG

Network Automation

1. Network Automation Introduction:

Network automation is the process of automating the configuration, management, testing, deployment, and operations of physical and virtual devices within a network.

Recently we start hearing netconf,yang for network automation. Simply Netconf is a protocol to configure/manage the device.

So far, how we configure the devices using.

1. CLI/UI

2. SNMP

Network Automation

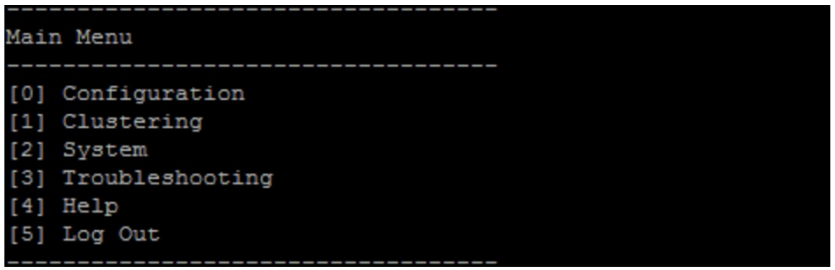
CLI

CLI scripting is the primary approach to making automated configuration changes to the network.

```
Switch>en
Switch#vlan data
% Warning: It is recommended to configure VLAN from config mode,
  as VLAN database mode is being deprecated. Please consult user
  documentation for configuring VTP/VLAN in config mode.

Switch(vlan)#vlan 10 Name DATA
VLAN 10 modified:
    Name: DATA
Switch(vlan)#vlan 20 Name VOICE
VLAN 20 modified:
    Name: VOICE
Switch(vlan)#apply
APPLY completed.
Switch(vlan)#exit
APPLY completed.
Exiting....
Switch#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Switch(config)#interface vlan 10
Switch(config-if)#ip address 10.1.10.10 255.255.255.0
Switch(config-if)#exit
Switch(config)#interface vlan 20
Switch(config-if)#ip address 10.1.20.10 255.255.255.0
Switch(config-if)#exit
Switch(config)#
Switch#wr
22:42:21: %SYS-5-CONFIG_I: Configured from console by console
Building configuration...
[OK]
Switch#
```

Network Automation



Using Perl, TCL/TK, Expect, Python scripts.

Lack of transaction management

Configuring a device maybe a complex task, involving multiple actions

Example: OSPF Protocol configuration in the Cisco router

https://study-ccna.com/ospf-configuration/

Usually these multiple actions can not be done partially, as this would leave the device in an undefined state. In case one action fails, we need to rollback. This requires extensive programming when transaction management is not supported and this is the case with CLI.

Network Automation

Syntax changes

Usually whenever a new device software is released, new CLI commands are added but unfortunately in some cases existing commands are modified or deleted. This means that every other software that was using CLI as an application programming interface (API) will fail

CLIs are designed to be used by humans and not an API for programmatic access.

Network Automation

SNMP

SNMP was developed to be used for both monitoring and management, i.e. it has the capability to write changes. So why is SNMP mostly used for fault and performance monitoring and not configuration changes?

No discovery

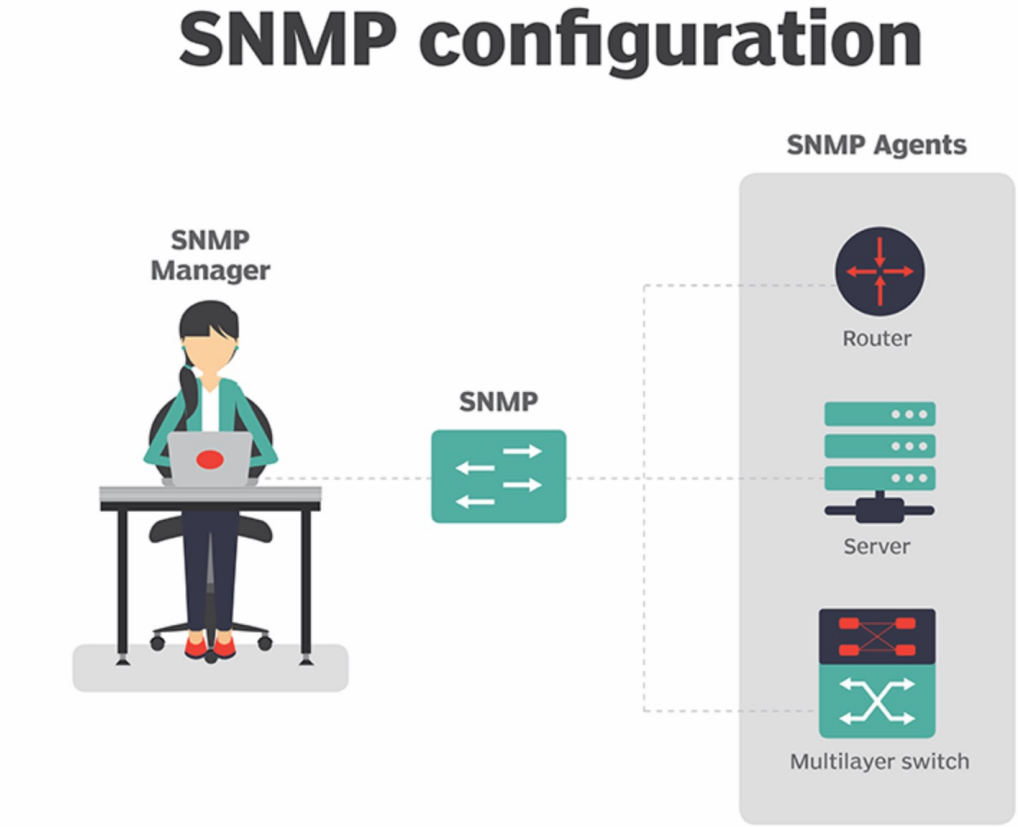
lack of a standard automatic discovery process that finds the (correct) MIB modules that the device is using.

This means that the discovery work must be done by the users or operators and as it is complex, configuration management with SNMP has been abandonded.

SNMP does not distinguish between the operational and configuration data stored within a device.

Its harder to program.

Network Automation



Network Automation

2. Netconf Key features

The key with NETCONF protocol is that it was designed to address the shortcomings of existing practices and protocols for configuration management including:

- Distinction between configuration and state data
- Multiple configuration data stores (candidate, running, startup)
- Configuration testing and validation support (validation, confirmed commit,rollback on error)
- Selective data retrieval with filtering
- Streaming and playback of event notifications (filter the notifications with date)
- Extensible procedure call mechanism (additional operations)

Network Automation

3. Current SDN/NFV Industry trends

- NETCONF is one of the key APIs used by SDN Controller on the Southbound to control network devices
- OpenDay Light, ONOS SDN Controller Supports Netconf in South bound Interface
- Automatic provisioning/orchestration

Automatic Service Deployment & Provisioning

NETCONF/YANG is a key enabler of the new paradigm of NFV,where services can be designed at a high level, independent of the complexities and device dependencies of the underlying infrastructure.

-Assembled "on the fly" from a range of virtualized functions from multiple vendors

-These services will be provisioned in minutes and their operation highly automated.

Network Automation

Zero touch provisioning:

Zero touch provisioning (ZTP) is a feature that allows the devices to be provisioned and configured automatically, eliminating most of the manual labor involved with adding them to a network.

Netconf plays a major role in the ZTP.

https://datatracker.ietf.org/doc/draft-ietf-netconf-zerotouch/

Network Automation

4. Current Network Automation Trends:

Figure 2: Network Automation Technology Stack

Layer	Examples
Software provisioning automation engines	Ansible, Chef, Salt, Puppet
Configuration templates	Juju, HOT, TOSCA
Configuration protocols	NETCONF
Encoding	XML, JSON
Modeling languages	UML, YANG
Programming languages	Python, Ruby

Source: Heavy Reading

Ansible, Chef, Puppet, Salt,

Netconf

1. NetConf Introduction

Ref: https://www.rfc-editor.org/rfc/rfc6241.txt

The NETCONF protocol defines a simple mechanism through which

- a network device can be managed,
- configuration data information can be retrieved,
- and new configuration data can be uploaded and manipulated.

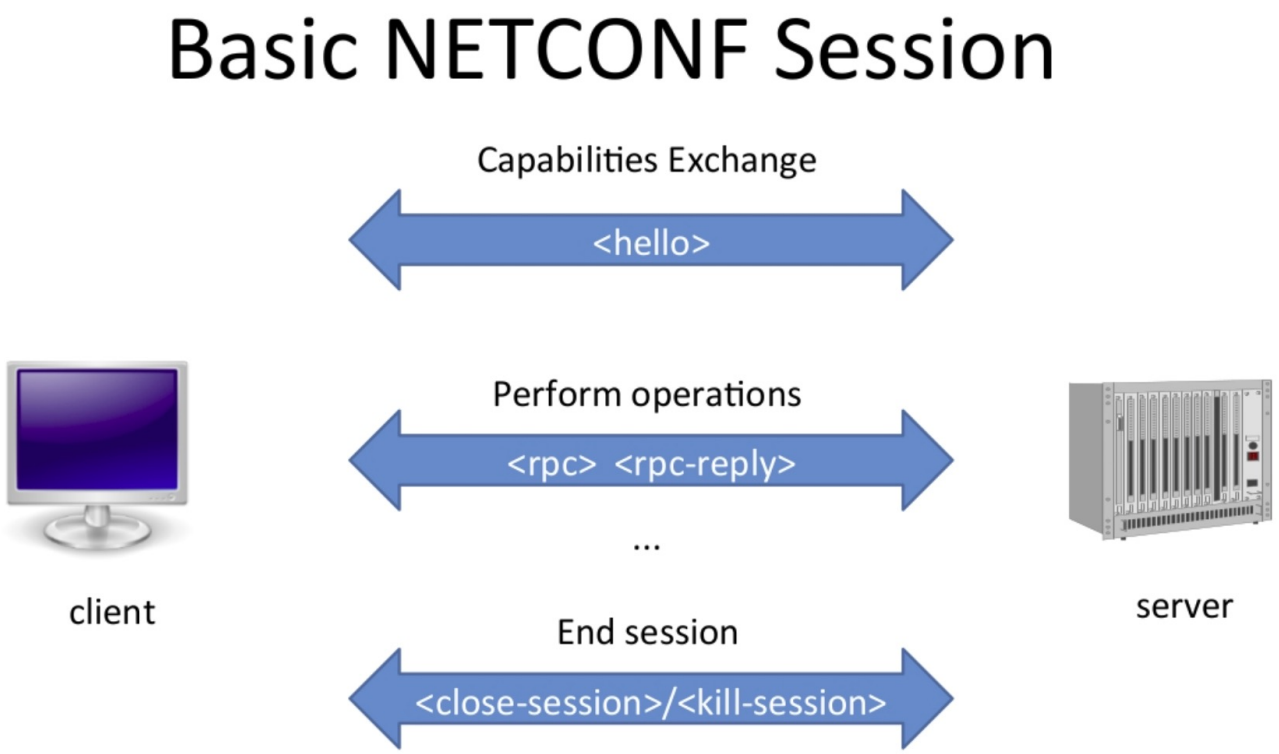
The protocol allows the device to expose application programming interface (API). Applications can use this straightforward API to send and receive full and partial configuration data sets.

The NETCONF protocol uses a remote procedure call (RPC) paradigm.

A client encodes an RPC in XML and sends it to a server using a secure, connection-oriented session.

The server responds with a reply encoded in XML.

Netconf


Basic NETCONF Session
Capabilities Exchange
<hello>
Perform operations
<rpc> <rpc-reply>
...
End session
<close-session>/<kill-session>
client
server

Netconf

The server is typically a network device. The client can be a script or application typically running as part of a network manager.

Messages:

1. Server & Client exchanges its capabilities in Hello Message (Session establishment)
2. Client sends RPC Message and get the RPC Reply Message

Netconf

HELLO Message:

```
<?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>
    <capability>urn:ietf:params:netconf:base:1.1</capability>
    <capability>urn:ietf:params:netconf:capability:writable-running:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:xpath:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:validate:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:validate:1.1</capability>
    <capability>urn:ietf:params:netconf:capability:rollback-on-error:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:notification:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:interleave:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:with-defaults:1.0?basic-mode=explicit&amp;also-
supported=report-all-tagged</capability>
    ................SKIPPED
  </capabilities>
<session-id>324</session-id>
</hello>
```

Netconf

Operations Message:

RPC

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="1">
  <get-config>
        <source>
            <running/>
        </source>
        <filter type="subtree">
             <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
             <hostname/>
            </native>
         </filter>
  </get-config>
</rpc>
```

Netconf

Response

```
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="1">
    <data>
        <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
            <hostname>csr1000v</hostname>
        </native>
    </data>
</rpc-reply>
```

Netconf

2. Transport

NetConf Port Number: TCP 830

1. Connection-Oriented Operation (TCP)
2. Authentication, Integrity, and Confidentiality

connections could be encrypted using Transport Layer Security (TLS) or Secure Shell (SSH) depending on the underlying protocol.

NETCONF connections MUST be authenticated. The transport protocol is responsible for authentication of the server to the client and authentication of the client to the server.

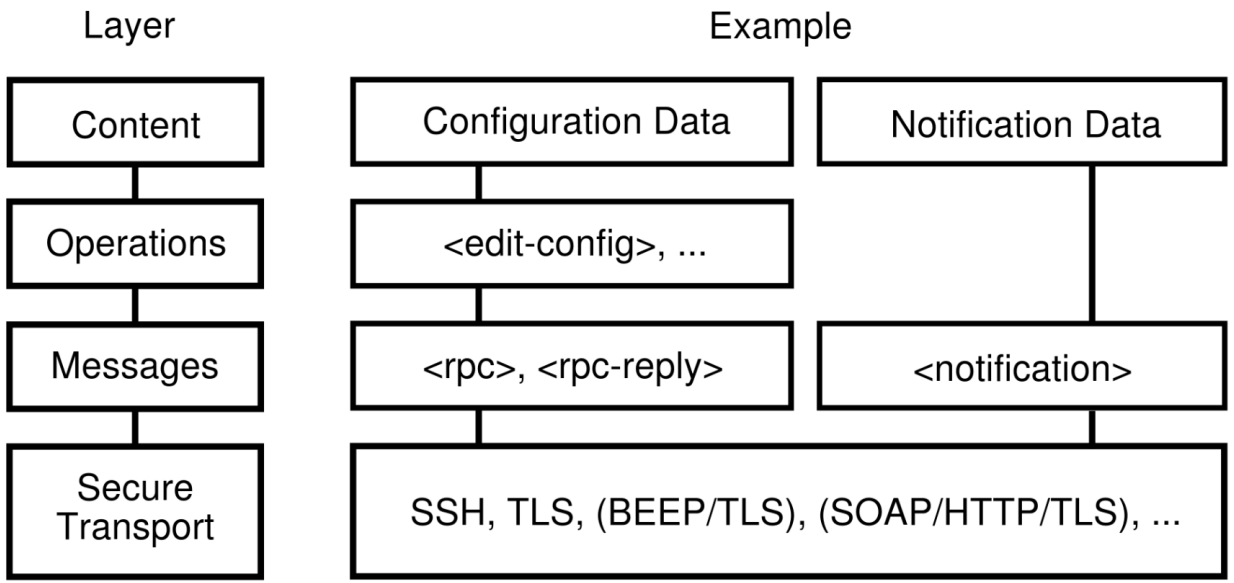
For example, a NETCONF server on a device that supports RADIUS might allow the use of RADIUS to authenticate NETCONF sessions

The access permissions of a given client, identified by its NETCONF username, are part of the configuration of the NETCONF server.

Netconf

3.Layers

NETCONF can be conceptually partitioned into four layers as shown in the below Figure.



Netconf

The Secure Transport layer provides a communication path between the client and server.

The Messages layer provides a simple, transport-independent framing mechanism for encoding RPCs and notifications.

The Operations layer defines a set of base protocol operations invoked as RPC methods with XML-encoded parameters.

The YANG data modeling language [RFC6020] has been developed for specifying NETCONF data models and protocol operations, covering the Operations and the Content layers of Figure .

Netconf

4. Messages

XML

XML serves as the encoding format for NETCONF, allowing complex hierarchical data to be expressed in a text format that can be read, saved, and manipulated with both traditional text tools and tools specific to XML.

All NETCONF protocol elements are defined in the following namespace:

```
**urn:ietf:params:xml:ns:netconf:base:1.0**
```

RPC

The NETCONF protocol uses an RPC-based communication model.

NETCONF peers use and elements to provide transport protocol-independent framing of NETCONF requests and responses.

Netconf

5.Datastores

NETCONF defines the existence of one or more configuration datastores and allows configuration operations on them.

Running DataStore

This is always present. The running configuration datastore holds the complete configuration currently active on the network device. Only one configuration datastore of this type exists on the device, and it is always present.

capability

> *urn:ietf:params:netconf:capability:writable-running:1.0*

Netconf

Startup DataStore

This is optional capability. stores the startup config. The device supports separate running and startup configuration datastores.

The startup configuration is loaded by the device when it boots. Operations that affect the running configuration will not be automatically copied to the startup configuration. An explicit operation from the to the is used to update the startup configuration to the current contents of the running configuration.

capability

> *urn:ietf:params:netconf:capability:startup:1.0*

Netconf

Candidate DataStore

This is optional capability. The candidate configuration is a full configuration data set that serves as a work place for creating and manipulating configuration data. Additions, deletions, and changes can be made to this data to construct the desired configuration data.

capability

> *urn:ietf:params:netconf:capability:candidate:1.0*

A operation MAY be performed at any time that causes the device's running configuration to be set to the value of the candidate configuration.

Netconf


Candidate
(:candidate)
<copy>
<commit>
Running
<copy>
Startup
(:startup)
Working copy to manipulate with no impact on current configuration
Complete and active configuration
Configuration loaded by the device at startup

Netconf Operations

1. Netconf Basic operations

The NETCONF protocol provides a small set of low-level operations to manage device configurations and retrieve device state information. The base protocol provides operations to retrieve, configure, copy, and delete configuration datastores.

Additional operations are provided, based on the capabilities advertised by the device(such as initialize the device etc).

- get
- get-config
- lock
- edit-config
- unlock
- close-session
- kill-session
- copy-config
- delete-config

Netconf Operations

a). get

Retrieve running configuration and device state information.

filter: This parameter specifies the portion of the system configuration and state data to retrieve.

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101">
  <get>
  </get>
</rpc>
```

b). get-config

Retrieve all or part of a specified configuration datastore.

source: Name of the configuration datastore being queried, such as .

filter: optional.we can specify the filter for the response.

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="1">
  <get-config>
    <source>
        <running/>
    </source>
  </get-config>
</rpc>
```

Netconf Operations

c). lock

The operation allows the client to lock the entire configuration datastore system of a device. Such locks are intended to be short-lived and allow a client to make a change without fear of interaction with other NETCONF clients, non-NETCONF clients (e.g., SNMP and command line interface (CLI) scripts), and human users.

```
<rpc message-id="101"
          xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
       <lock>
         <target>
           <running/>
         </target>
       </lock>
     </rpc>
```

Netconf Operations

d). edit-config

The operation updates the specified configuration to the specified target configuration datastore. In simplet term, configuring the device.

-**target:** Name of the configuration data store -**default-operation:** optional. merge(default),replace,none -**test-option**: optional. test-only, set, test-then-set -**error-option**: optional. stop-on-error, continue-on-error, rollback-on-error -**config:** A hierarchy of configuration data

```
<rpc message-id="101"
          xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
       <edit-config>
         <target>
           <running/>
         </target>
         <config>
           <top xmlns="http://example.com/schema/1.2/config">
             <interface>
               <name>Ethernet0/0</name>
               <mtu>1500</mtu>
             </interface>
           </top>
         </config>
       </edit-config>
     </rpc>
```

Netconf Operations

e). unlock

Description: The operation is used to release a configuration lock, previously obtained with the operation.

```
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <unlock>
    <target>
      <running/>
    </target>
  </unlock>
</rpc>
```

f). close-session

Request graceful termination of a NETCONF session.

The server will release any locks and resources associated with the session and gracefully close any associated connection

```
  <rpc message-id="101"
         xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
      <close-session/>
  </rpc>
```

Netconf Operations

g). kill-session

Description: Force the termination of a NETCONF session.

When a NETCONF entity receives a request for an open session, it will abort any operations currently in process, release any locks and resources associated with the session, and close any associated connections.

```
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <kill-session>
    <session-id>4</session-id>
  </kill-session>
</rpc>
```

Netconf Operations

2. Other Important Operations

a). Rollback

During Configuration of the device, If an error condition occurs such that an error severity element is generated, the server will stop processing the operation and restore the specified configuration to its complete state at the start of this operation. This option requires the server to support the :rollback-on-error capability

Netconf Operations

capability

> *urn:ietf:params:netconf:capability:rollback-on-error:1.0*

```
<rpc message-id="101"
          xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
       <edit-config>
         <target>
           <running/>
         </target>
         <error-option>rollback-on-error</error-option>
         <config>
           <top xmlns="http://example.com/schema/1.2/config">
             <interface>
               <name>Ethernet0/0</name>
               <mtu>100000</mtu>
             </interface>
           </top>
         </config>
       </edit-config>
     </rpc>
```

Netconf Operations

b) Validate Operation.

Validation consists of checking a complete configuration for syntactical and semantic errors before applying the configuration to the device.

capability

> *urn:ietf:params:netconf:capability:validate:1.1*

Netconf Operations

1.1 test-only" option in the parameter of the operation.

```
<rpc message-id="101"
          xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
       <edit-config>
         <target>
           <running/>
         </target>
         <test-option> test-only</test-option>
         <config>
           <top xmlns="http://example.com/schema/1.2/config">
             <interface>
               <name>Ethernet0/0</name>
               <mtu>100000</mtu>
             </interface>
           </top>
         </config>
       </edit-config>
     </rpc>
```

Netconf Operations

1.0 (Old format)

```
<rpc message-id="101"
          xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
       <validate>
         <source>
           <candidate/>
         </source>
       </validate>
</rpc>
```

Netconf Operations

3. Candidate datastore operations

capability

> *urn:ietf:params:netconf:capability:candidate:1.0*

a). Commit

To commit the candidate configuration as the device's new current configuration, use the operation.

If the device is unable to commit all of the changes in the candidate configuration store, then the running configuration MUST remain unchanged

```
<rpc message-id="101"
          xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
       <commit/>
     </rpc>
```

Netconf Operations

b). discard-changes

If the client decides that the candidate configuration is not to be committed, the operation can be used to revert the candidate configuration to the current running configuration.

```
    <rpc message-id="101"
           xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
        <discard-changes/>
      </rpc>
```

Netconf Operations

4. other datastore operations

capability

> *urn:ietf:params:netconf:capability:startup:1.0*

To save the startup configuration, use the operation to copy the configuration datastore to the configuration datastore.

a). copy-config

Create or replace an entire configuration datastore with the contents of another complete configuration datastore. If the target datastore exists, it is overwritten. Otherwise, a new one is created, if allowed.

```
<rpc message-id="101"
          xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
       <copy-config>
         <target>
           <startup/>
         </target>
         <source>
           <running/>
         </source>
       </copy-config>
</rpc>
```

Netconf Operations

b).delete-config

Description: Delete a configuration datastore. The configuration datastore cannot be deleted.

"startup" config can be removed.

```
      <rpc message-id="101"
           xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
        <delete-config>
          <target>
            <startup/>
          </target>
        </delete-config>
      </rpc>
```

Netconf Lab with ssh client

1. Introduction

This session demonstates the basic netconf operations using SSH client.

i) Device

Cisco CSR 1000v

Cisco provides free Lab environment(Device) called Devnet learning labs. In this Lab Cisco CSR Router 1000v is part of it, which can be used for Netconf demonstrations/practices.

Cisco IOS Sofware XE - 16.8.1

The device details can be collected from this below link

- https://devnetsandbox.cisco.com/RM/Diagram/Index/27d9747a-db48-4565-8d44-df318fce37ad?diagramType=Topology

Netconf Lab with ssh client

ii). Netconf clients

-Netconf over SSH - openssh client

-Python Netconf Client(ncclient)

-Ansible Netconf modules

-Opendaylight Netconf Connector

My System/VM OS is Ubuntu 18.04 .

Netconf Lab with ssh client

Netconf over SSH

Netconf protocol over SSH is standard available in RFC 6242.

we can use OpenSSH Client as Netconf Client for quick testing.

> *ssh username@IP -p portnumber -s netconf*

- https://tools.ietf.org/html/rfc6242
- https://www.juniper.net/documentation/en_US/junos/topics/task/operational/netconf-session-connecting-to-server.html
- https://community.calix.com/s/article/How-to-test-Netconf-messaging-to-an-E5E316F-using-SSH-and-Linux-1

Netconf Lab with ssh client

Python Netconf Client(ncclient)

Install the python netconf client

> *sudo pip install ncclient*

- https://github.com/CiscoDevNet/netconf-examples
- https://github.com/ncclient/ncclient

Ansible Netconf modules

Ansible netconf modume

- https://docs.ansible.com/ansible/2.5/modules/netconf_config_module.html
- https://docs.ansible.com/ansible/latest/modules/netconf_rpc_module.html

Opendaylight Netconf connector

The NETCONF southbound plugin is capable of connecting to remote NETCONF devices and exposing their configuration/operational datastores, RPCs and notifications as MD-SAL mount points. These mount points allow applications and remote users (over RESTCONF) to interact with the mounted devices.

https://docs.opendaylight.org/en/stable-oxygen/user-guide/netconf-user-guide.html

Netconf Lab with ssh client

i). Connection establishment

HELLO Message will be sent from both Client and Server with its capabilities.

> *ssh root@64.103.37.51 -p 10000 -s netconf*

```
$ ssh root@64.103.37.51 -p 10000 -s netconf
root@64.103.37.51's 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>
<capability>urn:ietf:params:netconf:base:1.1</capability>
<capability>urn:ietf:params:netconf:capability:writable-running:1.0</capability>
<capability>urn:ietf:params:netconf:capability:xpath:1.0</capability>
<capability>urn:ietf:params:netconf:capability:validate:1.0</capability>
<capability>urn:ietf:params:netconf:capability:validate:1.1</capability>
<capability>urn:ietf:params:netconf:capability:rollback-on-error:1.0</capability>
<capability>urn:ietf:params:netconf:capability:notification:1.0</capability>
<capability>urn:ietf:params:netconf:capability:interleave:1.0</capability>
<capability>urn:ietf:params:netconf:capability:with-defaults:1.0?basic-mode=explicit&amp;also-
supported=report-all-tagged</capability>
.................SKIPPED
</capabilities>
<session-id>324</session-id></hello>]]>]]
```

Netconf Lab with ssh client

Now we need to send the Client HELLO Message . Copy & Paste the below text in the SSH console.

```
<?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>
]]>]]>
```

Now connection establishment is over.

Netconf Lab with ssh client

ii). GET CONFIG Message

Reference: https://tools.ietf.org/html/rfc6241#section-7

Copy & Paste the below text in the SSH console.

```
<?xml version="1.0" ?>
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="1">
  <get-config>
    <source>
        <running/>
    </source>
  </get-config>
</rpc>
]]>]]>
```

Netconf Lab with ssh client

Reply

```
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="1"><data><native
xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native"><version>16.8</version><boot-start-marker/><boot-
end-marker/><banner><motd><banner>^C</banner></motd></banner><service><timestamps><debug>
......OUTPUT SKIPPED.............
</data></rpc-reply>]]>]]>
```

iii). GET Message

```
<?xml version="1.0" ?>
<rpc message-id="101"
          xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
       <get>
       </get>
     </rpc>
]]>]]>
```

Dont try. it will return plenty of stats and take lot of time to complete.

Netconf Lab with ssh client

iv). Edit config

Copy & Paste the below text in the SSH console.

```
<?xml version="1.0" ?>
     <rpc message-id="101"
          xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
       <edit-config>
         <target>
           <running/>
         </target>
         <config>
           <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
             <hostname>CISCOROUTER-SURESH</hostname>
            </native>
         </config>
       </edit-config>
     </rpc>
     ]]>]]>
```

Netconf Lab with ssh client

Reply

```
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101"><ok/></rpc-reply>]]>]]>
```

To verify the changes, query the getconfig with filter :

```
<?xml version="1.0" ?>
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="1">
  <get-config>
    <source>
        <running/>
    </source>
      <filter type="subtree">
             <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
             <hostname/>
             <version/>
          </native>
         </filter>
  </get-config>
</rpc>
]]>]]>
```

Netconf Lab with ssh client

v). Lock

```
<?xml version="1.0" ?>
<rpc message-id="101"
          xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
       <lock>
         <target>
           <running/>
         </target>
       </lock>
     </rpc>
]]>]]>
```

Reply

```
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101"><ok/></rpc-
reply>]]>]]>
```

Netconf Lab with ssh client

v). UnLock

```
<?xml version="1.0" ?>
<rpc message-id="101"
          xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
       <unlock>
         <target>
           <running/>
         </target>
       </unlock>
     </rpc>
]]>]]>
```

Reply:

```
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101"><ok/></rpc-
reply>]]>]]>
```

Netconf Lab with ssh client

vi). Edit config with Validate option

Case1:

```
<?xml version="1.0" ?>
     <rpc message-id="101"
          xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
       <edit-config>
         <target>
           <running/>
         </target>
         <test-option>test-only</test-option>
         <config>
           <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
             <hostname>CISCOROUTER-SURESH123</hostname>
          </native>
         </config>
       </edit-config>
     </rpc>
     ]]>]]>
```

Reply

```
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101"><ok/></rpc-
reply>]]>]]>
```

Netconf Lab with ssh client

Case2:

```
<?xml version="1.0" ?>
     <rpc message-id="101"
          xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
       <edit-config>
         <target>
           <running/>
         </target>
         <test-option>test-only</test-option>
         <config>
           <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
             <hostname>&-+***</hostname>
          </native>
         </config>
       </edit-config>
     </rpc>
     ]]>]]>
```

Netconf Lab with ssh client

vi). Edit config with Rollback option

```
<?xml version="1.0" ?>
     <rpc message-id="101"
          xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
       <edit-config>
         <target>
           <running/>
         </target>
         <error-option>rollback-on-error</error-option>
         <config>
           <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
             <hostname>CISCOROUTER-SURESH</hostname>
             <version>&-+***</version>
          </native>
         </config>
       </edit-config>
     </rpc>
     ]]>]]>
```

Subtree & XPath Filter

1. Subtree Filtering

XML subtree filtering is a mechanism that allows an application to select particular XML subtrees to include in the for a or operation. A small set of filters for inclusion, simple content exact-match, and selection is provided, which allows some useful, but also very limited, selection mechanisms.

Namespace Selection

A namespace is considered to match (for filter purposes) if the XML namespace associated with a particular node within the element is the same as in the underlying data model.

```
<filter type="subtree">
  <top xmlns="http://example.com/schema/1.2/config"/>
</filter>
```

Subtree & XPath Filter

i). Get Config

```
<?xml version="1.0" ?>
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="1">
  <get-config>
    <source>
        <running/>
    </source>
  </get-config>
</rpc>
]]>]]>
```

ii). Get Config with namespace filter

```
<?xml version="1.0" ?>
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="1">
  <get-config>
    <source>
        <running/>
    </source>
      <filter type="subtree">
        <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
        </native>
      </filter>
  </get-config>
</rpc>
]]>]]>
```

Subtree & XPath Filter

iii). Get Config with element filter

```
<?xml version="1.0" ?>
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="1">
  <get-config>
    <source>
        <running/>
    </source>
      <filter type="subtree">
        <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
        <hostname/>
        </native>
      </filter>
  </get-config>
</rpc>
]]>]]>
```

Subtree & XPath Filter

2. XPATH filter

The XPath capability indicates that the NETCONF peer supports the use of XPath expressions in the element

capability:

> *urn:ietf:params:netconf:capability:xpath:1.0*

iii). Get Config with XPATH filter

```
<?xml version="1.0" ?>
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="1">
  <get-config>
    <source>
        <running/>
    </source>
      <filter  xmlns:t="http://cisco.com/ns/yang/Cisco-IOS-XE-native" type="xpath"
select="/t:native/hostname">
      </filter>
  </get-config>
</rpc>
]]>]]>
```

Capabilities

1. Capabilities Introduction

https://tools.ietf.org/html/rfc6241

NETCONF allows a client to discover the set of protocol extensions supported by a server. These "capabilities" permit the client to adjust its behavior to take advantage of the features exposed by the device.

A NETCONF capability is a set of functionality that supplements the base NETCONF specification. A NETCONF capability is identified with a URI.

Capabilities defined in this document have the followingformat:

! urn:ietf:params:netconf:capability:{name}:1.x

where {name} is the name of the capability.

Capabilities augment the base operations of the device, describing both additional operations and the content allowed inside operations. The client can discover the server's capabilities and use any additional operations, parameters, and content defined by those capabilities.

Capabilities

Capabilities Exchange

Capabilities are advertised in messages sent by each peer during session establishment. When the NETCONF session is opened, each peer (both client and server) MUST send a element containing a list of that peer's capabilities

Each peer MUST send at least the base NETCONF capability, "urn:ietf:params:netconf:base:1.1".

```
<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:netconf:capability:startup:1.0
       </capability>
       <capability>
         http://example.net/router/2.3/myfeature
       </capability>
     </capabilities>
     <session-id>4</session-id>
   </hello>
```

Capabilities

2. Capabilities

A. Writable-Running Capability

The :writable-running capability indicates that the device supports direct writes to the configuration datastore. In other words, the device supports and operations where the configuration is the target.

> *urn:ietf:params:netconf:capability:writable-running:1.0*

B.Candidate Configuration Capability

The candidate configuration capability, :candidate, indicates that the device supports a candidate configuration datastore, which is used to hold configuration data that can be manipulated without impacting the device's current configuration.

> *urn:ietf:params:netconf:capability:candidate:1.0*

Capabilities

C.Rollback-on-Error Capability

This capability indicates that the server will support the "rollback-on-error" value in the parameter to the operation.

> *urn:ietf:params:netconf:capability:rollback-on-error:1.0*

D.Validate Capability

Validation consists of checking a complete configuration for syntactical and semantic errors before applying the configuration to the device.

> *urn:ietf:params:netconf:capability:validate:1.1*

Capabilities

E.Distinct Startup Capability

The device supports separate running and startup configuration datastores. The startup configuration is loaded by the device when it boots. Operations that affect the running configuration will not be automatically copied to the startup configuration. An explicit operation from the to the is used to update the startup configuration to the current contents of the running configuration. NETCONF protocol operations refer to the startup datastore using the element.

> *urn:ietf:params:netconf:capability:startup:1.0*

F. XPath Capability

The XPath capability indicates that the NETCONF peer supports the use of XPath expressions in the element.

> *urn:ietf:params:netconf:capability:xpath:1.0*

Capabilities

3. Analysis

HELLO Message will be sent from both Client and Server with its capabilities.

> *ssh root@64.103.37.51 -p 10000 -s netconf*

```
$ ssh root@64.103.37.51 -p 10000 -s netconf
root@64.103.37.51's 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>
<capability>urn:ietf:params:netconf:base:1.1</capability>
<capability>urn:ietf:params:netconf:capability:writable-running:1.0</capability>
<capability>urn:ietf:params:netconf:capability:xpath:1.0</capability>
<capability>urn:ietf:params:netconf:capability:validate:1.0</capability>
<capability>urn:ietf:params:netconf:capability:validate:1.1</capability>
<capability>urn:ietf:params:netconf:capability:rollback-on-error:1.0</capability>
<capability>urn:ietf:params:netconf:capability:notification:1.0</capability>
<capability>urn:ietf:params:netconf:capability:interleave:1.0</capability>
<capability>urn:ietf:params:netconf:capability:with-defaults:1.0?basic-mode=explicit&amp;also-
supported=report-all-tagged</capability>
.................SKIPPED
</capabilities>
<session-id>324</session-id></hello>]]>]]
```

Capabilities

Now we need to send the Client HELLO Message . Copy & Paste the below text in the SSH console.

```
<?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>
]]>]]>
```

In the capabilties, we can see the features supported by the Server as well as YANG Models supported.

Device discovery happens based on this information.

Additional/Custom RPCs(Operations)

1. Custom RPCs or Additional Operations

Custom RPCs or Additional Operations are defined in the YANG Models as RPC definitions. Netconf monitoring is one of the YANG Model, which supports get-schema rpc operation.

https://tools.ietf.org/html/rfc6022

xmlnamespace : urn:ietf:params:xml:ns:yang:ietf-netconf-monitoring

2. Get the schemas List

Make sure establish the session with hello.

```
<?xml version="1.0" ?>
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get>
    <filter type="subtree">
      <netconf-state xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-monitoring">
        <schemas/>
      </netconf-state>
    </filter>
  </get>
</rpc>
]]>]]>
```

The NETCONF server returns a list of schema available for retrieval.

Additional/Custom RPCs(Operations)

3. Get Schema RPC

Example1:

```
<?xml version="1.0" ?>
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<get-schema  xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-monitoring">
<identifier>Cisco-IOS-XE-platform</identifier>
</get-schema>
</rpc>
]]>]]>
```

Additional/Custom RPCs(Operations)

Example2:

```
<?xml version="1.0" ?>
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<get-schema  xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-monitoring">
<identifier>cisco-xe-ietf-ospf-deviation</identifier>
</get-schema>
</rpc>
]]>]]>
```

Example3:

```
<?xml version="1.0" ?>
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<get-schema  xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-monitoring">
<identifier>Cisco-IOS-XE-native</identifier>
</get-schema>
</rpc>
]]>]]>
```

Additional/Custom RPCs(Operations)

4. Save Config RPC

Further RPCs exeamples for Cisco XE Platform : https://www.cisco.com/c/en/us/support/docs/storage-networking/management/200933-YANG-NETCONF-Configuration-Validation.html

```
<?xml version="1.0" encoding="utf-8"?>
 <rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101">
   <cisco-ia:save-config xmlns:cisco-ia="http://cisco.com/yang/cisco-ia"/>
 </rpc>
]]>]]>
```

Reply

```
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101"><result
xmlns='http://cisco.com/yang/cisco-ia'>Save running-config successful</result>
</rpc-reply>]]>]]>
```

Notifications

1. Notifications

- NETCONF client indicates interest in receiving event notifications from a NETCONF server by creating a subscription to receive event notifications.
- The NETCONF server replies to indicate whether the subscription request was successful and, if it was successful, begins sending the event notifications to the NETCONF client as the events occur within the system.
- These event notifications will continue to be sent until either the NETCONF session is terminated or the subscription terminates for some other reason.

This operation initiates an event notification subscription that will send asynchronous event notifications to the initiator of the command until the subscription terminates.

capability

> *"urn:ietf:params:netconf:capability:notification:1.0"*

Notifications

2. Demo

My System/VM OS is Ubuntu 18.04 .

Here we use python netconf client to for listening the notifications,

Install the python netconf client

> *sudo pip install ncclient*

Notifications

Register the subscription for notifications and print the notification details when it occurs.

1. Run the notification script to continuously listening for notifications
2. open another terminal, and connect to the Netconf server using SSH Client
3. see the output in notification script.

```
$ python notifications.py
Waiting for next notification
<?xml version="1.0" encoding="UTF-8"?>
<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0"><eventTime>2019-06-
12T06:37:59.730733+00:00</eventTime>
<netconf-session-start xmlns='urn:ietf:params:xml:ns:yang:ietf-netconf-notifications'>
  <username>root</username>
  <session-id>238</session-id>
  <source-host>84.39.53.130</source-host>
</netconf-session-start>
</notification>
Waiting for next notification
```

2. Another terminal, connect to the netconf server using SSH client.

> *ssh root@64.103.37.51 -p 10000 -s netconf*