OBJECTIVES OF LABS

- Part 1: Build the Network and Verify Connectivity
- Part 2: Use a NETCONF Session to Gather Information
- Part 3: Use ncclient to Connect to NETCONF
- Part 4: Use ncclient to Retrieve Running Configuration (get-config)
- Part 5: Use ncclient to Edit Configuration (edit-config)

Netconf

Netconf is a network monitoring protocol which configure network devices like routers, switches programmatically. It removes configuration complexity, vendor specific commands and have better efficiency which is generally found in SNMP.

- **Data representation: XML**
- **Protocol used: SSH**
- **Alternative: RestConf**

➢ Features:

- Device configuration management (CRUD of devices)
- Transactional changes (support commit and rollback)
- Standardized and structured with Yang model

➢ Benefits:

- Automation-Ready (SDN and automation tool like ansible etc)
- Standardized (Yang models)
- Error Handling (validation and rollback)
- Security (due to SSH)

➢ Disadvantages:

- Complexity (XML and other learning curve)
- Performance overhead (XML And SSH compare to json and Rest features)
- Scalability (separate SSH connection for each SSH connection, which limits how many simultaneous sessions a server can manage.)

➢ Useful Commands:

- **get** - Retrieve operational data e.g., interface status.
- **get-config** - Retrieve configuration data.
- **edit-config** - Modify the device configuration.
- **commit** - Apply candidate configuration to the running state.
- **discard-changes** - Discard uncommitted changes.
- **lock and unlock** - Prevent concurrent edits during a session.
- **delete-config** - Delete a specified data store (e.g., candidate).
- **close-session**

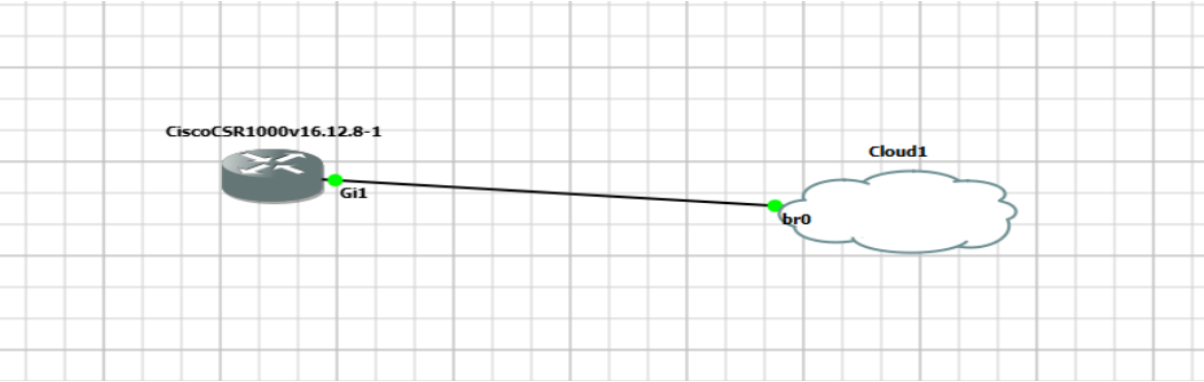
➢ What it solves:

- Automated Configuration
- Transaction Integrity
- Scalability
- Efficiency
- Example: In a large network, manually updating IP addresses on hundreds of routers is inefficient. NETCONF allows a centralized script to update all routers automatically using programs.

- **Tools Required:**
  - **GNS3**
  - **Router with capability to integrate with python (e.g., CSR 1000v)**
  - **Python with ncclient**
  - **GitHub**

Part 1: Build the Network and Verify Connectivity:

- **Topology:**
  - **Cisco csr 1000v connected to cloud with Giga Ethernet.**
  - **Cloud side is configured with br0.**



- **Setting up clock**

```
Router#clock set 01:01:01 11 dec 2014
Router#
Dec 11 01:01:01.012: %SYS-6-CLOCKUPDATE: System clock has been updated from 01:4
7:09 UTC Thu Dec 31 2015 to 01:01:01 UTC Thu Dec 11 2014, configured from consol
e by console.
Dec 11 01:01:01.018: %SMART_LIC-5-SYSTEM_CLOCK_CHANGED: Smart Agent for Licensin
g System clock has been changed
Router#show clock
01:01:12.166 UTC Thu Dec 11 2014
Router#
```

- **Interfaces:**
  - DHCP automatically assign Ip address to connected interface(gi1)

```
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#do sh ip int br
Interface              IP-Address      OK? Method Status                Protocol
GigabitEthernet1       172.20.0.20     YES DHCP   up                    up
GigabitEthernet2       unassigned      YES unset  down                  down
GigabitEthernet3       unassigned      YES unset  down                  down
GigabitEthernet4       unassigned      YES unset  down                  down
Router(config)#
```

➢ Configure SSH with cisco csr 1000v:

- Basic Configuration for password and banners

```
Router(config)#enable secret NMS@2025
Router(config)#banner motd "LAB  for Netconf using ncclient"
Router(config)#line console 0
Router(config-line)#login
% Login disabled on line 0, until 'password' is set
Router(config-line)#password NMS@2025
Router(config-line)#_
```

- Creating a username and provide privileges to user to all tasks.

```
Router(config)#username sanjog privilege 15 password NMS@2025
 WARNING: Command has been added to the configuration using a type 0 password. H
owever, type 0 passwords will soon be deprecated. Migrate to a supported passwor
d type
Router(config)#
Dec 11 01:07:42.490: %AAAA-4-CLI_DEPRECATED: WARNING: Command has been added to
the configuration using a type 0 password. However, type 0 passwords will soon b
e deprecated. Migrate to a supported password type
Router(config)#
```

- Now generate a RSA (Rivest-Shamir-Adleman) key pairs, which is essential for enabling secure communication protocols such as SSH (Secure Shell) and HTTPS.

```
Router(config)#hostname sanjog
sanjog(config)#crypto key generate rsa
The name for the keys will be: sanjog.sanjog.com
Choose the size of the key modulus in the range of 512 to 4096 for your
  General Purpose Keys. Choosing a key modulus greater than 512 may take
  a few minutes.

How many bits in the modulus [1024]: 1024
% Generating 1024 bit RSA keys, keys will be non-exportable...
[OK] (elapsed time was 0 seconds)

sanjog(config)#
Dec 11 01:11:00.564: %CRYPTO_ENGINE-5-KEY_ADDITION: A key named sanjog.sanjog.co
m has been generated or imported by crypto-engine
sanjog(config)#_
```


- Configure "ssh version 2" and then enter into virtual terminal.
- VTY lines control **remote access** (Telnet/SSH) to the Cisco device.
- "Transport input ssh" Restricts the transport protocol to **SSH** only.
- login local defines that Configures the router to authenticate remote users using local credentials

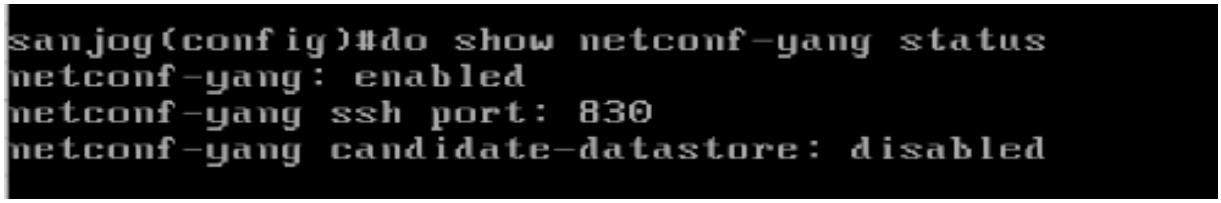
```
sanjog(config)#ip ssh version 2
sanjog(config)#line vty 0 15
sanjog(config-line)#transport input ?
  acercon   Remote console for ACE-based blade
  all       All protocols
  lat       DEC LAT protocol
  mop       DEC MOP Remote Console Protocol
  nasi      NASI protocol
  none      No protocols
  pad       X.3 PAD
  rlogin    Unix rlogin protocol
  ssh       TCP/IP SSH protocol
  telnet    TCP/IP Telnet protocol
  udptn     UDPTN async via UDP protocol

sanjog(config-line)#transport input ssh
sanjog(config-line)#login local
sanjog(config-line)#
```

- Verify User can access router remotely with SSH credentials.

Part 2: Use a NETCONF Session to Gather Information

- Enable Netconf in router:

- Verify router is enabled for Netconf



- Verify Netconf connection

Part 3: Use ncclient to Connect to NETCONF

- Using manager from ncclient library to connect with Router and verify with server capability.

```
my_host = "172.20.0.20"

if __name__ == "__main__":
    with manager.connect(
        host=my_host,
        port=830,
        username="sanjog",
        password="NMS@2025",
        hostkey_verify=False,
    ) as m:
        print("Connected to", my_host)
        for i in m.server_capabilities:
            print(i)
```

```
PS C:\Users\Administrator\Desktop\Code> python .\Netconf_using_ncclient.py
Connected to 172.20.0.20
urn:ietf:params:netconf:base:1.0
urn:ietf:params:netconf:base:1.1
urn:ietf:params:netconf:capability:writable-running:1.0
urn:ietf:params:netconf:capability:xpath:1.0
urn:ietf:params:netconf:capability:validate:1.0
urn:ietf:params:netconf:capability:validate:1.1
urn:ietf:params:netconf:capability:rollback-on-error:1.0
urn:ietf:params:netconf:capability:notification:1.0
urn:ietf:params:netconf:capability:interleave:1.0
urn:ietf:params:netconf:capability:with-defaults:1.0?basic-mode=explicit&also-supported=report-al
```

Part 4: Use ncclient to Retrieve Running Configuration (get-config)

- Python function for get_config

```python
def get_config_and_save(m, fileName):
    """
    Retrieves the running configuration from the device using NETCONF and saves it to a file.

    Parameters:
    m (ncclient.manager.Manager): The active NETCONF session manager.
    fileName (str): The name of the file where the configuration will be saved.

    Returns:
    None
    """
    c = m.get_config(source="running")
    with open(fileName, "w") as f:
        f.write(c.xml)
    print("Running configuration saved to", fileName)
```

- Generated xml file for get config

Part 5: Use ncclient to Edit Configuration (edit-config)

- Python script for edit config
- It creates a loopback address with user input for name ip and subnet mask

```python
def add_a_loopback(m):                                          Following Sanjogharinkhede
    """
    Prompts the user to enter loopback interface details and adds it to the running configuration.

    Parameters:
    m (ncclient.manager.Manager): The active NETCONF session manager.

    Returns:
    None
    """
    temp = f"""
    <config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
        <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces" xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-type">
            <interface>
                <name>{input("Enter loopback: ")}</name>
                <description>Loopback Added from Python by sanjog</description>
                <type>ianaift:softwareLoopback</type>
                <enabled>true</enabled>
                <ipv4 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip">
                    <address>
                        <ip>{input("Loopback IPv4 address: ")}</ip>
                        <netmask>{input("Loopback IPv4 netmask: ")}</netmask>
                    </address>
                </ipv4>
            </interface>
        </interfaces>
    </config>
    """
    c = m.edit_config(target="running", config=temp)
    print("Loopback interface added")
```

- User response on excuting the add_a_loopback(m) function:

```
Enter loopback: Loopback11
Loopback IPv4 address: 172.20.70.70
Loopback IPv4 netmask: 255.255.255.0
Loopback interface added
```

- Verify in Gns3

```
sanjog#sh ip int br
Interface              IP-Address      OK? Method Status                Protocol
GigabitEthernet1       172.20.0.20     YES DHCP   up                    up
GigabitEthernet2       unassigned      YES unset  down                  down
GigabitEthernet3       unassigned      YES unset  down                  down
GigabitEthernet4       unassigned      YES unset  down                  down
Loopback11             172.20.70.70    YES other  up                    up
sanjog#_
```