Sample Metro Deployment

This chapter covers the key components and sample end-to-end configuration involved in these Metro solution use cases:

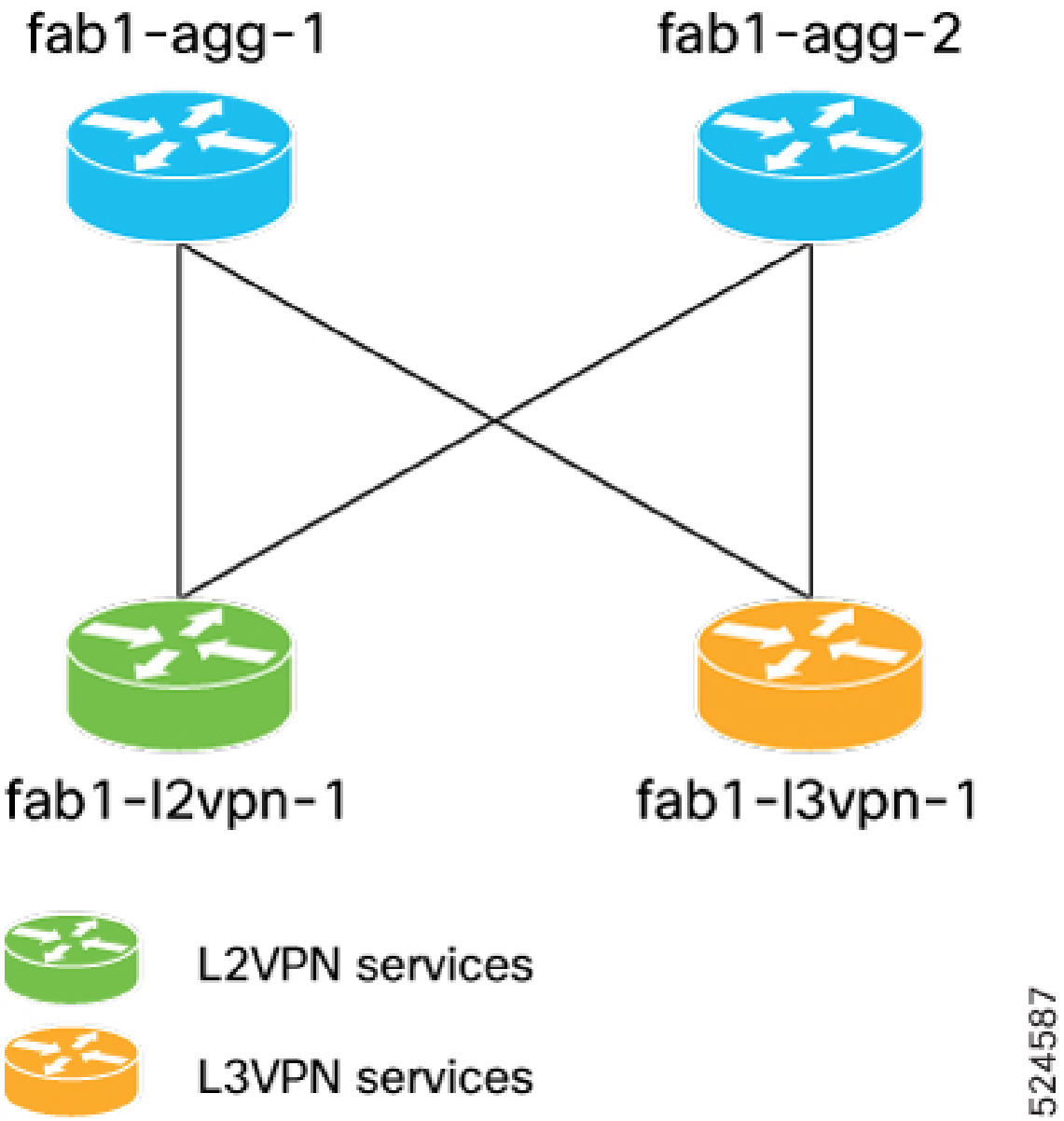
- Metro CX Edge Fabric Manager installation
- Metro CX Edge Fabric software life cycle management (FSLM)
- Metro CX Edge Fabric configuration template management
- Edge Protect DDoS deployment
- Cisco Routed PON deployment

Metro CX Edge Fabirc Manager installation

Sample Metro Edge Fabric deployment network

This is a sample topology diagram of Metro Edge Fabric deployment network.

Figure 1: Sample Metro Edge Fabric deployment network topology



The topology consists of two aggregation routers and two leaf nodes of different role types.

The aggregation routers in the fabric act as inline route reflectors. However, the route reflectors could be located elsewhere.

Control plane configuration

The table lists the protocols and parameters for control plane configuration.

Protocol	Parameters
BGP	ASN 100 IPv4 using separate loopback or route reflector (RR) IPv6 using separate loopback or RR Aggregation routers act as inline RR Aggregation routers have L3VPN and L2VPN address families enabled
IS-IS	Level 2 only SR-MPLS enabled SRv6 enabled TI-LFA enabled
Segment Routing	Flex-Algo 0 Flex-Algo 128 (low latency)
IPv4 IP Address Allocations	10.0.0.0/24 Flex-Algo 0 loopbacks 10.0.128.0/24 Flex-Algo 128 loopbacks
SR-MPLS SRGB Allocations	16000-32000 SRGB 16000-16999 Algo 0 17000-17999 Algo 128
SRv6 IP Address Allocations	2001:DC::/24 Base SRv6 global block 2001:DC00::/32 Flex-Algo 0 2001:DC80::/32 Flex-Algo 128

Additional configuration

The table lists the configuration required for additional elements.

Element	Parameter
Route Reflectors	10.0.0.100 10.0.0.101
SR-PCE	10.0.0.200 10.0.0.201
SNMP Trap Server	10.0.0.250

Address resource allocation

The table lists the address resource allocation for each node.

Node	IPv4 loopback	SR-MPLS SIDs	SRv6 locator	IPv6 loopback
fab1-agg-1	10.0.0.1/32 algo 0 10.0.1.1/32 algo 128	16001 algo 0 17001 algo 128	2001:DC00:0001::/48 algo 0 2001:DC80:0001::/48 algo 128	2001:DC00:0001::1/128
fab1-agg-2	10.0.0.2/32 algo 0 10.0.1.2/32 algo 128	16002 algo 0 17002 algo 128	2001:DC00:0002::/48 algo 0 2001:DC80:0002::/48 algo 128	2001:DC00:0002::1/128
fab1-l2vpn-1	10.0.0.3/32 algo 0 10.0.1.3/32 algo 128	16003 algo 0 17003 algo 128	2001:DC00:0003::/48 algo 0 2001:DC80:0003::/48 algo 128	2001:DC00:0003::1/128
fab1-l3vpn-1	105.0.0.4/32 algo 0 105.0.1.4/32 algo 128 10.0.0.4/32 algo 0 10.0.1.4/32 algo 128	16004 algo 0 17004 algo 128	2001:DC00:0004::/48 algo 0 2001:DC80:0004::/48 algo 128	2001:DC00:0004::1/128

CX Fabric Manager NSO solution components

This table lists the various NSO packages and its purpose.

NSO package	Version	Purpose
resource-manager	4.2.3	Define IP address pools for allocation through ZTP
ztp	3.7	CX ZTP package used for onboarding
os-upgrade	1.2	CX OS upgrade package used for software life-cycle management
edge-fabric-manager	4.3	Main Edge Fabric Management application used for fabric or role definition, config template application, and compliance actions
cisco-iosxr-cli-7.52	7.52.2	NSO CLI NED

Install CX Fabric Manager package

Follow these steps to install CX Fabric Manager package.

Procedure

Step 1 Download standard resource-manager package from CCO.

Step 2 Download CX Fabric Manager, CX ZTP, and CX OS upgrade packages

Step 3 Copy all packages to the packages directory relevant to your specific system or local installation of NSO. Example: /var/opt/nso/packages/

Step 4 Expand the CX NSO packages within the package directory so that the FM package can access the template directory.

Step 5 Verify packages.

Example:

```
Router# show packages package oper-stat


                           PACKAGE
NAME                       VERSION  UP
-----------------------------------------------
cisco-iosxr-cli-7.52       7.52.2   X
cisco-iosxr-nc-7.11        7.11     X
edge-fabric-manager        4.3      X
os-upgrade                 1.2      X
resource-manager           4.2.3    X
ztp                        3.7      X
```

Manage CX Fabric Manager template

Templates are defined as NSO XML config files and are located in the *templates* directory for the expanded package post installation.

Fabric, role, and interface templates used by the Fabric Manager are standard NSO configuration templates stored with the FM Package.

Role templates are referenced in the role definition as the filename without the .xml file extension.

Note Adding or removing templates from the *templates* directory requires a package reload.

View available templates

Use the **ll** command to display the available templates in the templates directory.

```
root@nso-edge-fabric-test-1:/var/opt/ncs/packages/edge-fabric-manager/templates# ll
total 36
drwxrwxr-x 2 cisco cisco 4096 Jun 12 10:12 ./
drwxrwxr-x 6 cisco cisco 4096 Jun 21 13:56 ../
-rw-rw-r-- 1 cisco cisco 2289 May 7 14:24 day1-leaf-l2vpn-v1.xml
-rw-rw-r-- 1 cisco cisco 2113 May 7 14:24 day1-leaf-l2vpn.xml
-rw-rw-r-- 1 cisco cisco 3214 May 9 09:03 day1-leaf-l3vpn.xml
```

```
-rw-rw-r-- 1 cisco cisco 1585 May 7 14:24 delete-leaf-interface-connection.xml
-rw-rw-r-- 1 cisco cisco 532 May 7 14:24 fabric-generic.xml
-rw-rw-r-- 1 cisco cisco 3665 May 7 14:24 leaf-interface-connection-v1.xml
-rw-rw-r-- 1 cisco cisco 3369 May 7 14:24 leaf-interface-connection.xml

root@nso-edge-fabric-test-1:/var/opt/ncs/packages/edge-fabric-manager/templates#
```

Base templates

The base configuration templates apply to all routers in the fabric.

Primary loopback IP and SNMP templates

These are examples of templates defining the primary Loopback IP and SNMP.

Primary loop:

```
<devices xmlns="http://tail-f.com/ns/ncs">
  <device>
    <name>{$DEVICE}</name>
    <config>
      <interface xmlns="http://tail-f.com/ned/cisco-ios-xr">
        <Loopback>
          <id>0</id>
          <description>Loopback for L2VPN Leaf</description>
          <ipv4>
            <address>
              <ip>{$LOOPBACK_IP}</ip>
              <mask>255.255.255.255</mask>
            </address>
          </ipv4>
        </Loopback>
       </interface>
    </config>
  </device>
</devices>
```

SNMP:

```
  <devices xmlns="http://tail-f.com/ns/ncs">
    <device>
      <name>{$DEVICE}</name>
      <config>
        <snmp-server xmlns="http://tail-f.com/ned/cisco-ios-xr">
          <ifmib>
            <ifalias>
              <long/>
            </ifalias>
            <stats>
              <cache/>
            </stats>
          </ifmib>
          <packetsize>4096</packetsize>
          <ifindex>persist</ifindex>
          <host>
            <address>10.0.0.250</address>
            <type>traps</type>
            <community-string>${SNMPTRAP_COMM}</community-string>
            <version>2c</version>
            <udp-port>1062</udp-port>
          </host>
          <community>
```

```
            <name>${SNMP_RW_COMM}</name>
            <RW/>
          </community>
          <community>
            <name>${SNMP_RO_COMM}</name>
            <RO/>
          </community>
          <location>"Fabric-A Location"</location>
          <trap-source>
            <MgmtEth>0/RP0/CPU0/0</MgmtEth>
          </trap-source>
        </snmp-server>
      </config>
    </device>
  </devices>
```

PTP configuration

```
<devices xmlns="http://tail-f.com/ns/ncs">
    <device>
      <name>{$DEVICE}</name>
      <config>
        <ptp xmlns="http://tail-f.com/ned/cisco-ios-xr">
          <clock>
            <domain>24</domain>
            <profile>
              <name>g.8275.1</name>
              <clock-type>T-BC</clock-type>
            </profile>
          </clock>
          <profile>
            <name>timeTransmitter</name>
            <multicast>
              <target-address>
                <ethernet>01-80-C2-00-00-0E</ethernet>
              </target-address>
            </multicast>
            <transport>ethernet</transport>
            <sync>
              <frequency>16</frequency>
            </sync>
            <clock>
              <operation>two-step</operation>
            </clock>
            <announce>
              <frequency>8</frequency>
            </announce>
            <delay-request>
              <frequency>16</frequency>
            </delay-request>
          </profile>


          <profile>
            <name>timeReceiver</name>
            <multicast>
              <target-address>
                <ethernet>01-80-C2-00-00-0E</ethernet>
              </target-address>
            </multicast>
            <transport>ethernet</transport>
            <announce>
              <timeout>5</timeout>
```

```
            <frequency>8</frequency>
          </announce>
          <delay-request>
            <frequency>16</frequency>
          </delay-request>
        </profile>
        <physical-layer-frequency/>
        <log>
          <servo>
            <events/>
          </servo>
          <best-master-clock>
            <changes/>
          </best-master-clock>
        </log>
      </ptp>
    </config>
  </device>
</devices>
```

IS-IS configuration

```
<devices xmlns="http://tail-f.com/ns/ncs">
    <device>
      <name>{$DEVICE}</name>
      <config>
        <router xmlns="http://tail-f.com/ned/cisco-ios-xr">
          <isis>
            <tag>
              <name>CORE</name>
            <is-type>level-2-only</is-type>
              <net>
                <id>{$ISIS_NET}</id>
              </net>
              <flex-algo>
                <id>128</id>
                <metric-type>
                  <delay/>
                </metric-type>
                <advertise-definition/>
              </flex-algo>
              <distribute>
                <link-state>
                  <instance-id>{$BGP_LS_INSTANCE_ID}</instance-id>
                </link-state>
              </distribute>
              <log>
                <adjacency>
                  <changes/>
                </adjacency>
                <pdu>
                  <drops/>
                </pdu>
              </log>
              <lsp-refresh-interval>65000</lsp-refresh-interval>
            <max-lsp-lifetime>65535</max-lsp-lifetime>

               <address-family>
                <ipv4>
                  <unicast>
                    <metric-style>wide</metric-style>
                   <maximum-paths>32</maximum-paths>
                    <segment-routing>
```

```
                      <mpls/>
                    </segment-routing>
                  </unicast>
                </ipv4>
                <ipv6>
                  <unicast>
                    <metric-style>wide</metric-style>
                    <maximum-paths>32</maximum-paths>
                  </unicast>
                </ipv6>
              </address-family>
              <interface>
                <name>Loopback0</name>
                <interface-type>passive</interface-type>
                <address-family>
                  <ipv4>
                    <unicast>
                      <prefix-sid>
                        <index>{$SR_SID_INDEX_ALGO_0}</index>
                      </prefix-sid>
                      <prefix-sid-algorithm>
                        <prefix-sid>
                          <algorithm>
                            <id>128</id>
                            <absolute>{$SR_SID_ABS_ALGO_128}</absolute>
                       </algorithm>
                       </prefix-sid>
                       </prefix-sid-algorithm>
                    </unicast>
                  </ipv4>
                  <ipv6>
                    <unicast/>
                  </ipv6>
                </address-family>
              </interface>
         </tag>
        </isis>
        </router>
      </config>
    </device>
 </devices>
```

Leaf templates

L2VPN leaf template: BGP

```
<devices xmlns="http://tail-f.com/ns/ncs">
    <name>{$DEVICE}</name>
    <config>
      <router xmlns="http://tail-f.com/ned/cisco-ios-xr">
        <bgp>
          <bgp-no-instance>
            <id>{$BGP_ASN}</id>
            <nsr/>
            <bgp>
              <router-id>{$BGP_ROUTER_ID}</router-id>
              <graceful-restart/>
            </bgp>
            <ibgp>
              <policy>
                <out>
                  <enforce-modifications/>
```

```
            </out>
          </policy>
        </ibgp>

        <address-family>
          <l2vpn>
            <evpn/>
          </l2vpn>
        </address-family>
        <neighbor-group>
          <name>SvRR-EVPN</name>
          <remote-as>{$BGP_ASN}</remote-as>
          <update-source>
            <Loopback>0</Loopback>
          </update-source>
          <address-family>
            <l2vpn>
              <evpn/>
            </l2vpn>
          </address-family>
        </neighbor-group>
        <neighbor>
          <id>10.0.0.100</id>
          <use>
            <neighbor-group>SvRR-EVPN</neighbor-group>
          </use>
        </neighbor>
        <neighbor>
          <id>10.0.0.101</id>
          <use>
            <neighbor-group>SvRR-EVPN</neighbor-group>
          </use>
        </neighbor>
      </bgp-no-instance>
    </bgp>
  </router>
 </config>
</device>
</devices>
```

L3VPN leaf template: BGP

```
<devices xmlns="http://tail-f.com/ns/ncs">
    <name>{$DEVICE}</name>
      <config>
       <router xmlns="http://tail-f.com/ned/cisco-ios-xr">
        <bgp>
          <bgp-no-instance>
            <id>{$BGP_ASN}</id>
            <nsr/>
            <bgp>
              <router-id>{$BGP_ROUTER_ID}</router-id>
              <graceful-restart/>
            </bgp>
            <ibgp>
              <policy>
                <out>
                  <enforce-modifications/>
                </out>
              </policy>
            </ibgp>

            <address-family>
```

```
            <vpnv4>
              <unicast/>
            </vpnv4>
            <vpnv6>
              <unicast/>
            </vpnv6>
          </address-family>
          <neighbor-group>
            <name>SvRR-L3VPN</name>
            <remote-as>{$BGP_ASN}</remote-as>
            <update-source>
              <Loopback>0</Loopback>
            </update-source>
            <address-family>
              <vpnv4>
                <unicast/>
              </vpnv4>
            </address-family>
          </neighbor-group>
          <neighbor>
            <id>10.0.0.102</id>
            <use>
              <neighbor-group>SvRR-L3VPN</neighbor-group>
            </use>
          </neighbor>
          <neighbor>
            <id>10.0.0.103</id>
            <use>
              <neighbor-group>SvRR-L3VPN</neighbor-group>
            </use>
          </neighbor>
        </bgp-no-instance>
      </bgp>
    </router>
  </config>
 </device>
</devices>
```

View loaded templates

Use the **show packages package edge-fabric-manager templates** command on the router CLI to view the loaded templates.

```
Router-cisco@ncs#show packages package edge-fabric-manager templates
templates [ day1-leaf-l2vpn day1-leaf-l2vpn-v1 day1-leaf-l3vpn
delete-leaf-interface-connection fabric-base-isis fabric-base-loopback fabric-base-ptp
fabric-base-snmp fabric-generic leaf-interface-connection leaf-interface-connection-v1
spine-interface-connection ]
```

Example of Fabric and role definition

This example defines a fabric with the ID, *fab1*, with a base template, *fab1-base-template*, already defined.

The *xr-leaf-l2vpn* role and its properties are shown. Multiple templates and their associated variables are defined as part of the role definition. The role also defines an interface-template with its own variables which can be used to perform operations such as adding the interface to a specific ISIS instance or assigning a PTP profile.

```
fabrics fabric-id fab1
 fabric-description "Example Fabric"
 fabric-tags        "test;cisco"
```

```
 fabric-template-id fab1-base-template
 device-role device-role-name xr-leaf-l2vpn device-model N540-24Z8Q2C-M
  topology-role     leaf
  role-templates role-template-id leaf-l2vpn-bgp-1
   role-template-variables name BGP_ASN
   !
   role-template-variables name BGP_ROUTER_ID
   !
  !
  role-templates role-template-id leaf-l2vpn-isis-v1
   role-template-variables name BGP_LS_INSTANCE_ID
   !
   role-template-variables name ISIS_NET
   !
   role-template-variables name SR_SID_INDEX_ALGO_0
   !
   role-template-variables name SR_SID_ABS_ALGO_128
   !
  !
  interface-template interface-template-id fab1-interface-template
  interface-template interface-template-variables name MTU
  !
  interface-template interface-template-variables name PTP_PROFILE
  target-os-version 7.11.2
 !
!
```

Onboard Fabric

In this deployment we onboard the spine devices manually and onboard the leaf devices using ZTP. The leaf devices are automatically onboarded into the fabric with a specific role during the ZTP process.

Follow these steps to onboard Fabric.

Procedure

Step 1 Define spine fabric role.

Example:

```
fabrics fabric-id fab1
 device-role device-role-name xr-fabric-spine device-model NCS-5501-SE
  topology-role     spine
  role-templates role-template-id fabric-base-isis
   role-template-variables name BGP_LS_INSTANCE_ID
   !
   role-template-variables name ISIS_NET
   !
   role-template-variables name SR_SID_ABS_ALGO_128
   !
   role-template-variables name SR_SID_INDEX_ALGO_0
   !
  !
  role-templates role-template-id fabric-base-loopback
   role-template-variables name LOOPBACK_IP
   !
  !
  role-templates role-template-id fabric-base-ptp
  !
```

```
  role-templates role-template-id fabric-base-snmp
  !
  interface-template interface-template-id spine-interface-connection
  target-os-version 7.11.2
```

Step 2 Define DHCP device for the spine device.

Example:

```
devices dhcp-devices FOC2120R22S
 device-host-name   fab1-spine-2
 fabric-id          fab1
 geo-location       west
 nso-authgroup      fabric
 nso-device-group   ALL-ACCESS
 device-role-name   xr-fabric-spine
 mgmt-ip            192.168.1.192
 subnet-mask        24
 resource-pool-name fabric-lab
 device-model       NCS-5501-SE
 template-variable-values template-type role
  variables name BGP_LS_INSTANCE_ID
   value 100
  !
  variables name ISIS_NET
   value 49.0001.0105.0000.0002.00
  !
  variables name LOOPBACK_IP
   value 10.0.0.2
  !
  variables name SR_SID_INDEX_ALGO_0
   value 2
  !
  variables name SR_SID_ABS_ALGO_128
   value 17002
 !
!
```

These are some of the attributes in this example:

- Device type: NCS-5501-SE
- Device serial number: FOC2120R22S
- DCHP device: added with the key being the device serial number
- Four base templates applied to the spine device: PTP, IS-IS, SNMP, and the loopback interface.

  IS-IS and the loopback have device-specific variables defined. PTP and SNMP use hard-coded values in the template itself.

Step 3 Onboard the spine device.

a) Onboard the device into NSO.

Example:

```
devices device fab1-spine-2
 address   192.168.1.192
 authgroup fabric
```

```
 device-type cli ned-id cisco-iosxr-cli-7.52
 state admin-state unlocked
```

b) Onboard the device into spine role.

Example:

```
Router:cisco@ncs# edge-fabric-actions onboard-device-into-fabric-apply-day1 devices { device-name
 fab1-spine-2 device-model NCS-5501-SE fabric-id fab1 device-role-name xr-fabric-spine serial-number
 FOC2120R22S geo-location west }
devices {
    device-name fab1-spine-2
    fabric-id fab1
    status Completed
    response
                        Dry-run location is:
/home/nso/temp/dry-run-output/fabric-device-onboard-fab1_fab1-spine-2_2024-09-15T18:47:36.189545.txt
}
```

c) Verify spine device onboarding.

Example:

Interface loopback:

```
Router:fab1-spine-2#show run int lo0
Sun Sep 15 17:21:56.428 UTC
interface Loopback0
 description Loopback interface
 ipv4 address 105.0.0.2 10.0.0.2 255.255.255.255
!
```

SNMP:

```
Router:fab1-spine-2#show run snmp
Sun Sep 15 17:26:20.468 UTC
snmp-server host 105.0.0.250 traps version 2c public udp-port 1062
snmp-server community cisco RW
snmp-server community public RO
snmp-server community private RW
snmp-server location "Fabric-A Location"
snmp-server packetsize 4096
snmp-server trap-source MgmtEth0/RP0/CPU0/0
snmp-server ifmib ifalias long
snmp-server ifindex persist
snmp-server ifmib stats cache
```

IS-IS:

```
Router:fab1-spine-2#show run router isis
Sun Sep 15 17:25:57.621 UTC
router isis CORE
 is-type level-2-only
 net 49.0001.0105.0000.0002.00
 distribute link-state instance-id 100
 log adjacency changes
 log pdu drops
 lsp-refresh-interval 65000
 max-lsp-lifetime 65535
 address-family ipv4 unicast
  metric-style wide
  maximum-paths 32
```

```
 segment-routing mpls
!
address-family ipv6 unicast
 metric-style wide
 maximum-paths 32
!
flex-algo 128
 metric-type delay
 advertise-definition
!
interface Loopback0
 passive
 address-family ipv4 unicast
  prefix-sid index 2
  prefix-sid algorithm 128 absolute 17002
```

PTP:

```
Router:fab1-spine-2#show run ptp
Sun Sep 15 17:25:20.506 UTC
ptp
 clock
  domain 24
  profile g.8275.1 clock-type T-BC
 !
 profile timeReceiver
  multicast target-address ethernet 01-80-C2-00-00-0E
  transport ethernet
  announce timeout 5
  announce frequency 8
  delay-request frequency 16
 !
 profile timeTransmitter
  multicast target-address ethernet 01-80-C2-00-00-0E
  transport ethernet
  sync frequency 16
  clock operation two-step
  announce frequency 8
  delay-request frequency 16
 !
 physical-layer-frequency
 log
  servo events
  best-master-clock changes
```

Step 4 Define L2VPN leaf fabric role.

Example:

```
fabrics fabric-id fab1
 device-role device-role-name xr-fabric-leaf-l2vpn device-model N540-12Z20G-SYS
  topology-role     leaf
  role-templates role-template-id fabric-base-isis
   role-template-variables name BGP_LS_INSTANCE_ID
   !
   role-template-variables name ISIS_NET
   !
   role-template-variables name SR_SID_ABS_ALGO_128
   !
   role-template-variables name SR_SID_INDEX_ALGO_0
   !
  !
  role-templates role-template-id fabric-base-loopback
```

```
   role-template-variables name LOOPBACK_IP
   !
  !
  role-templates role-template-id fabric-base-ptp
  !
  role-templates role-template-id fabric-base-snmp
  !
  role-templates role-template-id fabric-leaf-l2vpn-bgp
   role-template-variables name BGP_ASN
   !
   role-template-variables name BGP_ROUTER_ID
   !
  !
  interface-template interface-template-id spine-interface-connection
  target-os-version 7.11.2
 !
!
```

Step 5 Define DHCP device for L2VPN leaf node.

Example:

```
devices dhcp-devices FOC2430PL4Z
 device-host-name   fab1-l2vpn-1
 fabric-id          fab1
 geo-location       west
 nso-authgroup      fabric
 nso-device-group   ALL-ACCESS
 device-role-name   xr-l2vpn
 mgmt-ip            192.168.1.64
 subnet-mask        24
 resource-pool-name fabric-lab
 device-model N540-12Z20G-SYS
 template-variable-values template-type role
  variables name BGP_ASN
   value 100
  !
  variables name BGP_ROUTER_ID
   value 10.0.0.3
  !
  variables name BGP_LS_INSTANCE_ID
   value 100
  !
  variables name ISIS_NET
   value 49.0001.0105.0000.0003.00
  !
  variables name LOOPBACK_IP
   value 10.0.0.3
  !
  variables name SR_SID_INDEX_ALGO_0
   value 3
  !
  variables name SR_SID_ABS_ALGO_128
   value 17003
 !
!
```

This step is same as the previous example of spine device with the addition of the BGP template.

The DCHP device is added with the key being the device serial number. The role template variables are populated with device-specific values

Step 6 Configure DHCP day0 configuration.

Example:

```
!! IOS XR
hostname ###HOST_NAME###
logging console disable
!
username lab
 group root-lr
 group cisco-support
 secret 5 $1$CcGF$EzBAkyycnbZFt4QRF16I20
!
grpc
 no-tls
 port 57344
 address-family ipv4
!
interface MgmtEth0/RP0/CPU0/0
 ipv4 address ###MGMT_IP### ###MGMT_MASK###
 description To MGMT network
 no shutdown
!
router static
 address-family ipv4 unicast
  0.0.0.0/0 192.168.1.1
 !
!
line console
 exec-timeout 0 0
!
line default
 exec-timeout 0 0
!
fpd auto-upgrade enable
netconf-yang agent
 ssh
!
lldp
!
ssh client source-interface MgmtEth0/RP0/CPU0/0
ssh server logging
ssh timeout 120
ssh server rate-limit 600
ssh server session-limit 100
ssh server v2
ssh server vrf default
ssh server netconf vrf default
```

The DHCP day0 configuration applies baseline configuration to the device so that further configuration can take place.

This file is located on the DCHP server and referenced as part of the DHCP role settings in *dhcpd.conf*. The variables are populated by the CX ZTP process.

Step 7 Initiate ZTP

Example:

```
Router:fab1-l2vpn-1#ztp initiate debug dhcp4 management verbose
```

How zero touch provisioning works

These stages describe the zero touch provisioning (ZTP) process.

1. The system initiates DHCP request on the management interface.
2. DHCP responds with interim management IP address, location of Day 0 configuration for the device, and the CX ZTP Python script.
3. The system applies Day 0 configuration to the device for management connectivity and user setup.
4. The user executes CX ZTP onboarding script on the router which in turn performs these operations:
   a. Assigns permanent management IP from NSO resource pool
   b. Onboards device into NSO with the specific hostname and allocated IP address
   c. Executes automated fabric onboarding to onboard the device into the specified fabric with the appropriate role templates applied

Verify Fabric onboarding

Procedure

Verify fabric onboarding.

Example:

```
Router:cisco@ncs# show fabrics devices
FABRIC                                                                                                  GEO
 OPER
ID       DEVICE NAME    DEVICE ROLE NAME      DEVICE MODEL       INTERFACE TEMPLATE ID    LOCATION
 STATE  ROLE TEMPLATE ID
-------------------------------------------------------------------------------------------------------------------
fab1     fab1-l2vpn-1   xr-fabric-leaf-l2vpn  N540-12Z20G-SYS-A  interface-template       west
 READY fabric-base-isis

       fabric-base-loopback

       fabric-base-ptp

       fabric-base-snmp

       fabric-leaf-l2vpn-bgp
         fab1-spine-2   xr-fabric-spine       NCS-5501-SE        interface-template       west
 READY fabric-base-isis

       fabric-base-loopback

       fabric-base-ptp

       fabric-base-snmp
```

Create Fabric connections

Once devices are onboarded into NSO and the fabric, you can then create device connections between them.

Connections contain endpoints. A single endpoint is useful for configuring a spine device performing in-band ZTP for downstream leaf devices.

Procedure

Step 1 Create connection between interfaces on both the routers.

Example:

```
Router-cisco@ncs#edge-fabric-actions populate-fabric-connections fabric-connection-details { fabric-id
 fab1 fabric-connections { connection-id spine1_leaf1 connection-type fabric endpoints { endpoint
fab1-spine-2 interface TenGigE0/0/0/32 ip-address 10.120.1.1/24 } endpoints { endpoint fab1-l2vpn-1
 interface TenGigE0/0/0/32 ip-address 10.120.1.50/24 } } } commit-type commit
```

In this example, we create a connection between TenGigE0/0/0/22 on both the *fab1-spine-2* and *fab1-l2vpn-1* routers.

The default commit-type is a **dry-run** to show the configuration without deploying to the device. Use the **commit** commit-type to commit the configuration to NSO and deploy to the devices.

Step 2 Verify fabric connections.

Example:

```
Router-cisco@ncs# show fabrics fabric-connections

FABRIC   CONNECTION ID       TYPE        ENDPOINT       INTERFACE        IP ADDRESS

-------------------------------------------------------------------------------------------

fab1     spine1_leaf1        fabric      fab1-l2vpn-1   TenGigE0/0/0/32  10.120.1.50/24

                                         fab1-spine-2   TenGigE0/0/0/32  10.120.1.1/24
```

Metro CX Edge Fabric software life cycle management

This section highlights applying the steps required to perform a software upgrade utilizing the Crosswork Workflow Manager workflows which determine whether the device complies with the intended target OS version. If the device is not in compliance, we remediate the condition by upgrading the device software.

For a brief on software life cycle management, see Fabric software lifecycle management.

Perform prerequisite configurations for software upgrade

In Metro Release 1.0 software upgrades are performed using the CX OS Upgrade package. This is a comprehensive NSO-based service used to handle device upgrades for NX OS, IOS XE, and IOS XR OS.

Before you begin

You must define the device types and supported versions in the OS Upgrade package prior to performing upgrades.

In this example, we are upgrading a device of type NCS540L

Procedure

Step 1 Define the supported upgrade paths between software versions.

Example:

```
os-upgrade-service lookup-data upgradePathLookup image-version-mapping cisco-ios-xr
 entries NCS540L 7.11.1 24.4.1.37I
 !
 entries NCS540L 24.4.1.37I 7.11.1
 !
!

os-upgrade-service lookup-data platform-lookup platform-mapping cisco-ios-xr
 platform NCS540L
  model-keywords NCS540L
   device-model  NCS540L
   upgrade-reload-time 600
  !
  firmware-upgrade-enabled false
 !
!

os-upgrade-service lookup-data os-upgrade-vars image-vars image-version cisco-ios-xr NCS540L 24.4.1.37I

 vars SYSTEM_IMAGE
  var-value ncs540l-x64-24.4.1.37I.iso
 !
 vars target-version
  var-value 24.4.1.37I
 !
!

os-upgrade-service lookup-data version-image-lookup image-version-mapping cisco-ios-xr
 entries NCS540L 7.11.1
  image-filename [ ncs540l-x64-7.11.1.iso ]
 !
 entries NCS540L 24.4.1.37I
  image-filename [ ncs540l-x64-24.4.1.37I.iso ]
 !
!
```

In this example, we are upgrading from 7.11.1 to 24.4.1.37I.

Step 2 Define Fabric Manager role.

Example:

```
fabrics fabric-id metro10-fabric1
 device-role device-role-name xr-leaf device-model N540-24Q8L2DD-SYS
  topology-role     leaf
  target-os-version 24.4.1.37I
```

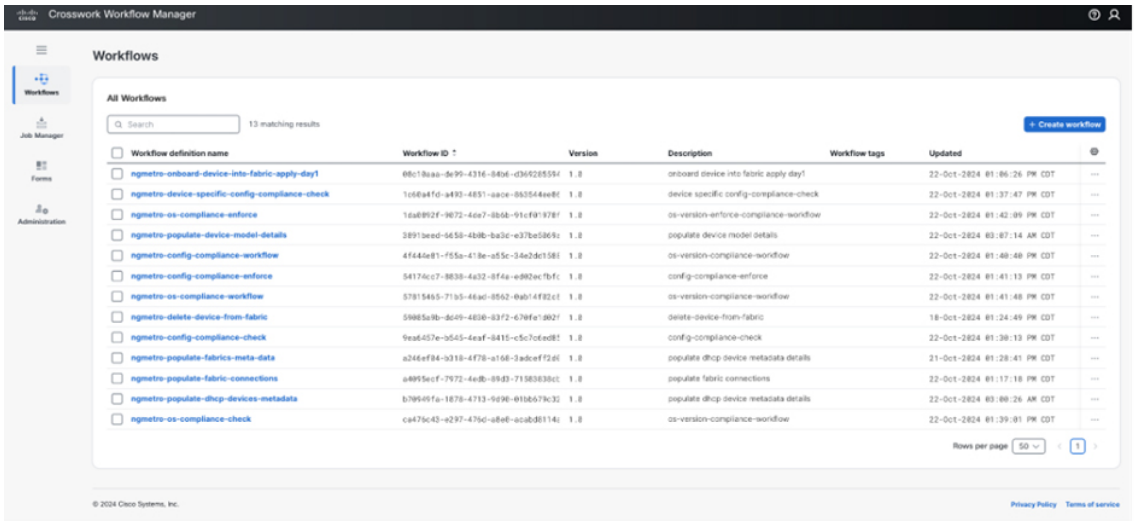
!
!

The intended version of the operating system is defined in the role definition. The device-model is used as a key and as an input to the CX OS upgrade service to determine the correct files and methodology for upgrading the device.

Crosswork Workflow Manager

Crosswork Workflow Manager is a flexible tool that is used to create customized network workflows. Workflows can be hierarchical in nature, with parent workflows with specific inputs used to drive child workflows to both collect data as well as execute actions against resources. In this use case, we are using CWM to execute flows to check the current software version of the device against an intended version defined by the fabric role.

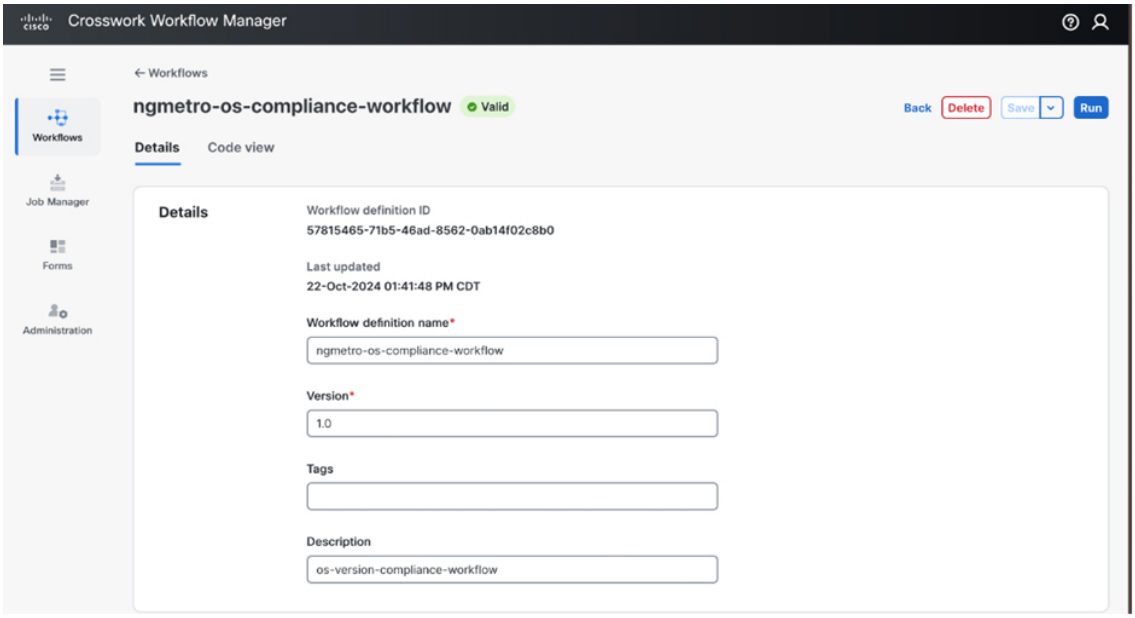
Figure 2: Crosswork Workflow Manager



Device OS compliance workflow

We use `ngmetro-os-compliance-workflow`, a compound workflow for performing device OS compliance check and remediation.

Figure 3: Device OS compliance workflow



These stages describe the device OS compliance workflow.

1. The compound workflow, `ngmetro-os-compliance-workflow`, first calls the `ngmetro-os-compliance-check` workflow to check for non-compliant devices.

   The `ngmetro-os-compliance-check` workflow uses these attributes as the input:

   ```
   ngmetro-os-compliance-check
       input:
       {
        "fabricId" : "metro10-fabric1",
        "nsoResource" : "METRO-NSO"
       }
   ```

2. The `ngmetro-os-compliance-check` workflow induces the OS compliance check function that is part of the Fabric Manager service.

3. When that task is executed, the `ngmetro-os-compliance-check` workflow returns the status of the devices in the fabric.

   In this case, the device `fabric-leaf3` is found as non-compliant.

   ```
   {
       "eventId": "21",
       "eventTime": "2024-10-22T18:39:16.152371499Z",
       "eventType": "WorkflowExecutionCompleted",
       "taskId": "2097494",
       "workflowExecutionCompletedEventAttributes": {
         "result": {
           "payloads": [
             {
               "metadata": {
                 "encoding": "json/plain"
               },
   ```

```
            "data": {
              "Data": {
                "checkComplianceResult": [
                  {
                    "device-name": "fabric-leaf3",
                    "os-compliant-status": false,
                    "response": " Device fabric-leaf3 OS version is not compliant.
Existing version is: 7.11.2.17I and target version is: 24.4.1."
                  }
                ],
                "fabricId": "metro10-fabric1",
                "nsoResource": "METRO-NSO"
              }
            }
          }
        ]
      },
      "workflowTaskCompletedEventId": "20"
    }
  }
]
```


4. The compound workflow runs the `ngmetro-os-compliance-enforce` workflow to perform the proper upgrades based on the target software version defined in the fabric roles.

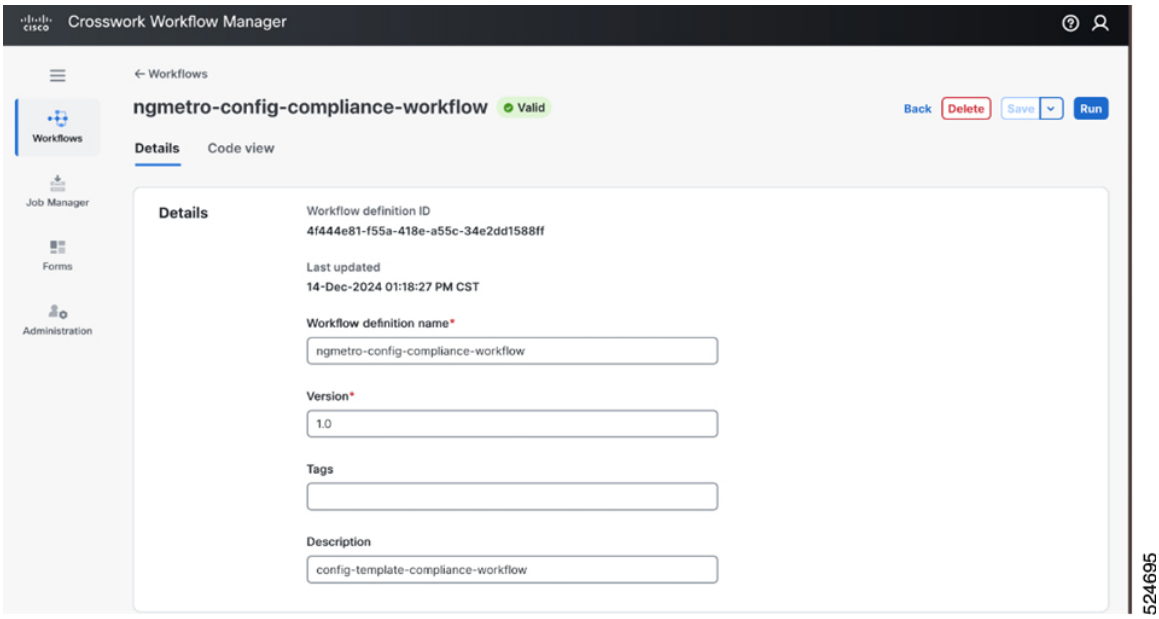
What's Next

The next step in the workflow is to remediate the device and upgrade it to the target version specified. For conciseness we are omitting the details of that step, but it utilizes the CX OS upgrade configuration mentioned earlier to upgrade the device matching type NCS540L with the appropriate Cisco IOS XR software version 24.4.1.

Metro CX Edge Fabric configuration template compliance

Perform configuration template compliance check and remediation

Figure 4: Crosswork Workflow Manager: configuration compliance workflow



Similar to the OS compliance workflow, we use a compound workflow in CWM to perform a configuration template compliance check and remediation:

- `ngmetro-config-compliance-workflow` workflow: to check the current assigned template in the Fabric Manager definition against what has been deployed on the node, and
- `ngmetro-config-compliance-enforce-workflow`: for remediation.

For a brief on configuration template management, see the Configuration compliance section.

Follow these steps to run the compound workflow for configuration template compliance check and remediation.

Procedure

Step 1 Run the `ngmetro-config-compliance-check` workflow with the appropriate fabric as input.

Example:

```
"data": {
              "status": 200,
              "data": {
                "edge-fabric-manager:output": {
                  "fabrics": [
                    {
```

```
                        "devices": [
                          {
                            "device-name": "fabric-leaf3",
                            "interface-compliant-status": false,
                            "response": "Device fabric-leaf3 Role-template is compliant. Device
fabric-leaf3 Interface-template is not compliant. Existing Interface-template is:
leaf-interface-connection and latestInterface-template is: interconnect-template.",
                            "role-compliant-status": true
                          }
                        ],
                        "fabric-id": "metro10-fabric1"
                      }
                    ]
                  }
                }
```


This step checks all the nodes in the Fabric against their intended templates vs. the runtime templates and returns whether the templates defined for the device and role match the runtime templates.

In this example the role template is compliant, but the interface template has been changed from `leaf-interface-connection` to `interconnect-template` and is no longer compliant.

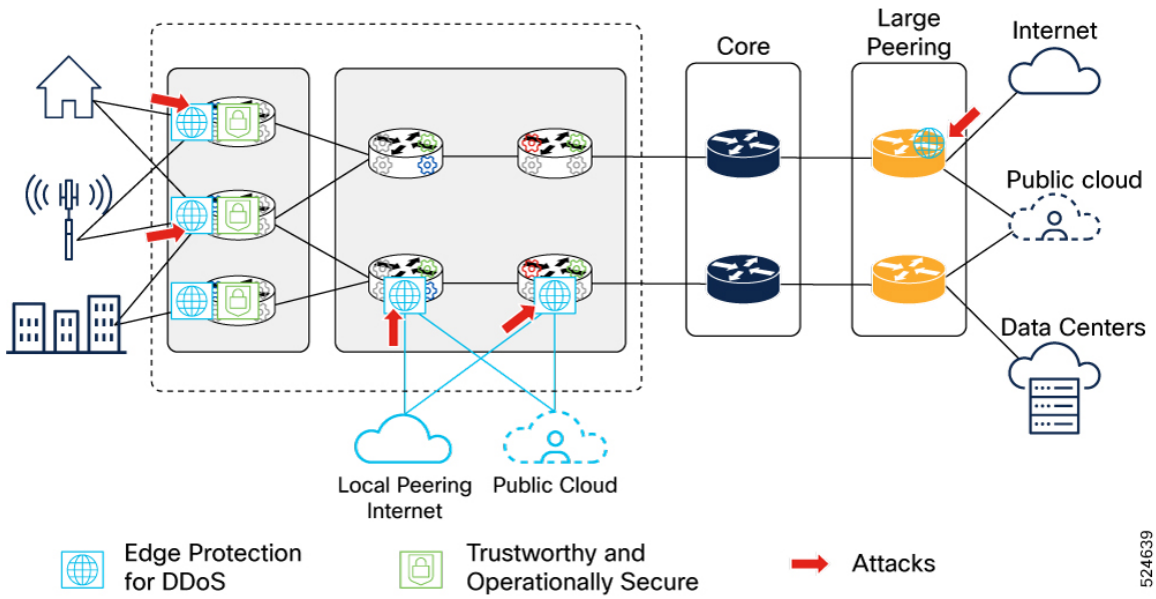
Step 2 Once you find devices with non-compliant templates, run the remediation workflow—`ngmetro-config-compliance-enforce-workflow`—with the appropriate fabric as input.

This step applies the proper configuration templates to devices which have been changed.

Edge Protect DDoS deployment

Once the Edge Fabric is deployed, you can deploy additional value-added services to the Fabric. Edge Protect DDoS is an innovative solution allowing service providers to deploy DDoS protection at the very edge of the network. This helps not only protect end services from DDoS and other security threats but also helps detect and mitigate attacks originating from within a service provider network.

Figure 5: High-level view of Edge Protect DDoS deployment topology



Edge Protect components

The Edge Protect solution consists of these components:

- Edge DDoS Protection Controller: This component runs as a VM on a virtualization infrastructure. The controller is responsible for configuring and deploying detectors, configuring protected objects, and monitoring detected attacks and active mitigations.
- Edge DDoS Protection Detector: This component runs as a third-party application on IOS-XR routers, providing a distributed detection and mitigation engine. There is a single detector deployed to each router.

Figure 6: Detector lists

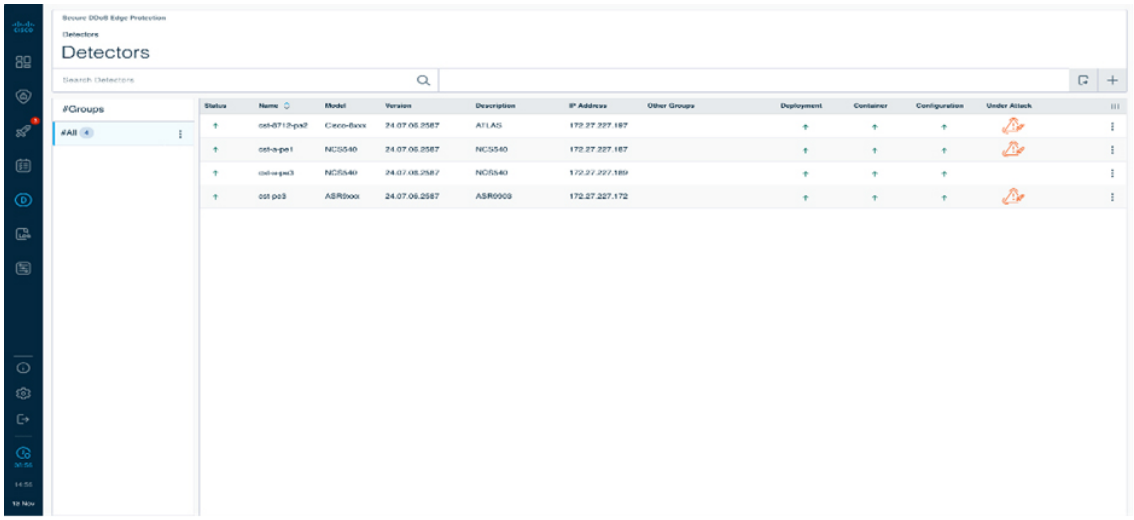
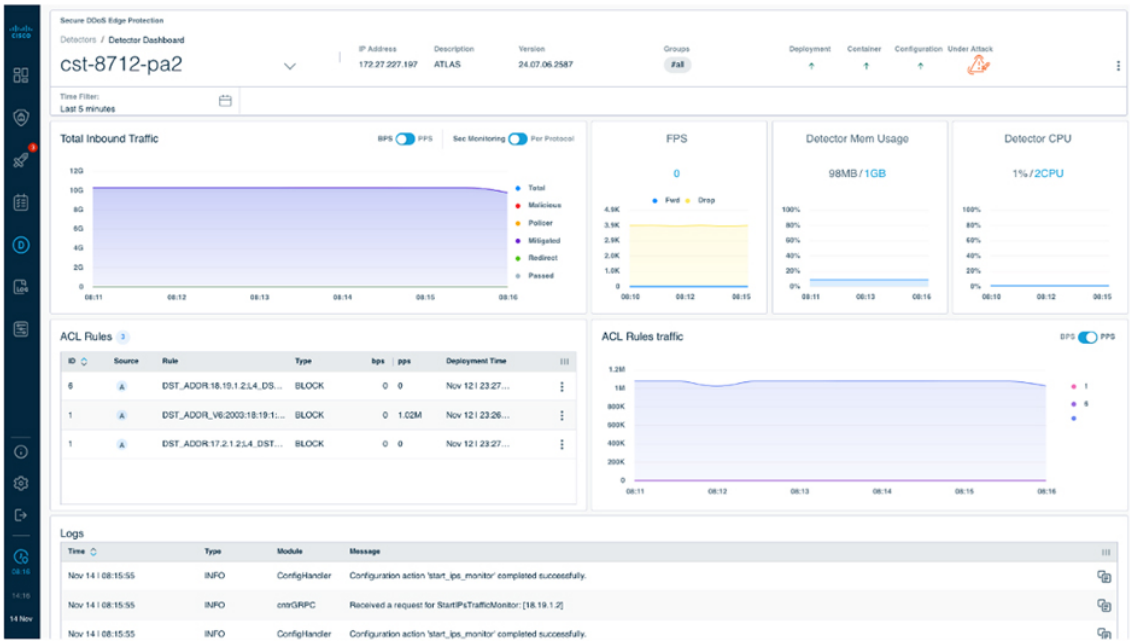


Figure 7: Detector details



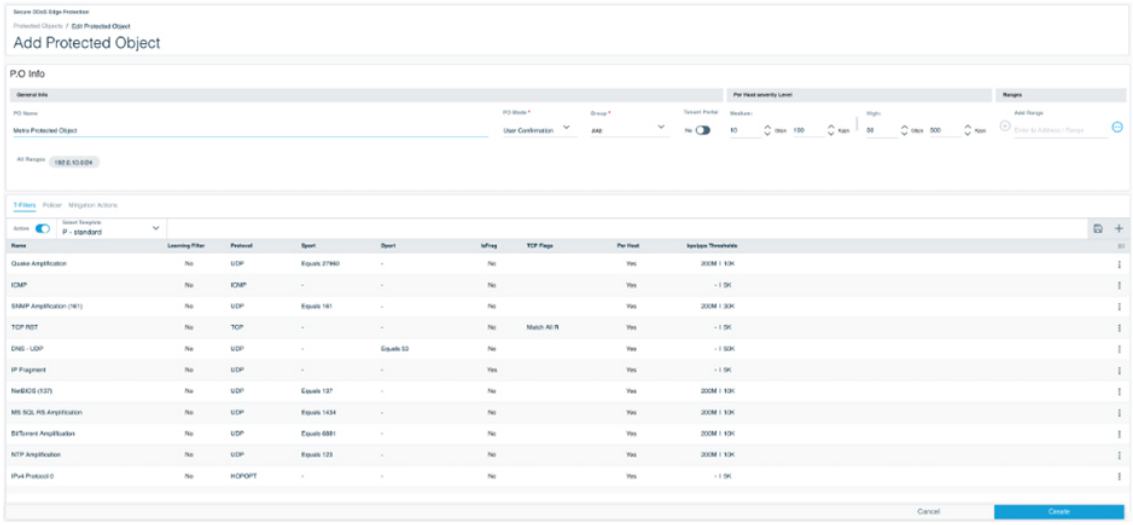
Edge Protect operation

The Edge Protect operation involves Edge Protect DDoS detection and mitigation. The detection phase uses advanced analytics and examinations to identify specific amplification and application layer attacks. The mitigation phase is performed by deploying either user-driven or automated ACLs to distributed nodes to limit the attack impact.

Edge Protect DDoS detection

Detecting attacks requires having flow-level visibility of traffic traversing the router. In the Edge Protect solution Netflow data is encoded into Google Protobuf format and then consumed by the on-board detection engine. Deployment of the appropriate Netflow configuration is done automatically by the controller when a router acting as a detector is onboarded. Standard or user-defined detection templates are used as a basis for detecting DDoS traffic patterns. These are applied to protected objects, which can apply to an entire device, or a subset of IP addresses based on configuration. Protected objects are then configured with policing and additional mitigation actions.

Figure 8: Edge detect projected objects

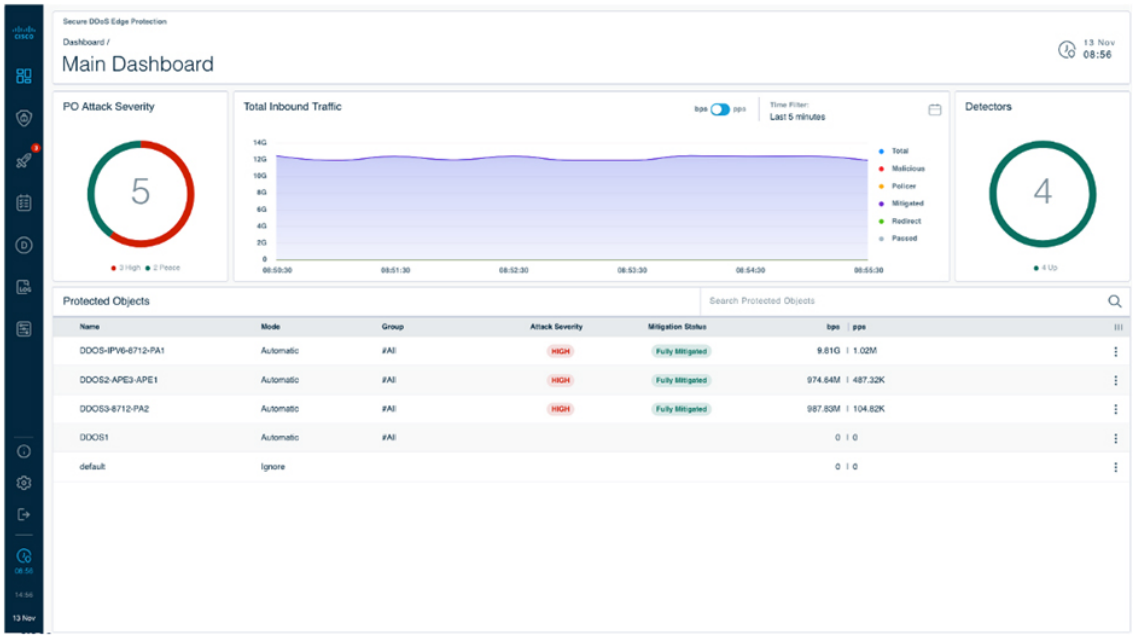


524698

Edge Protect DDoS mitigation

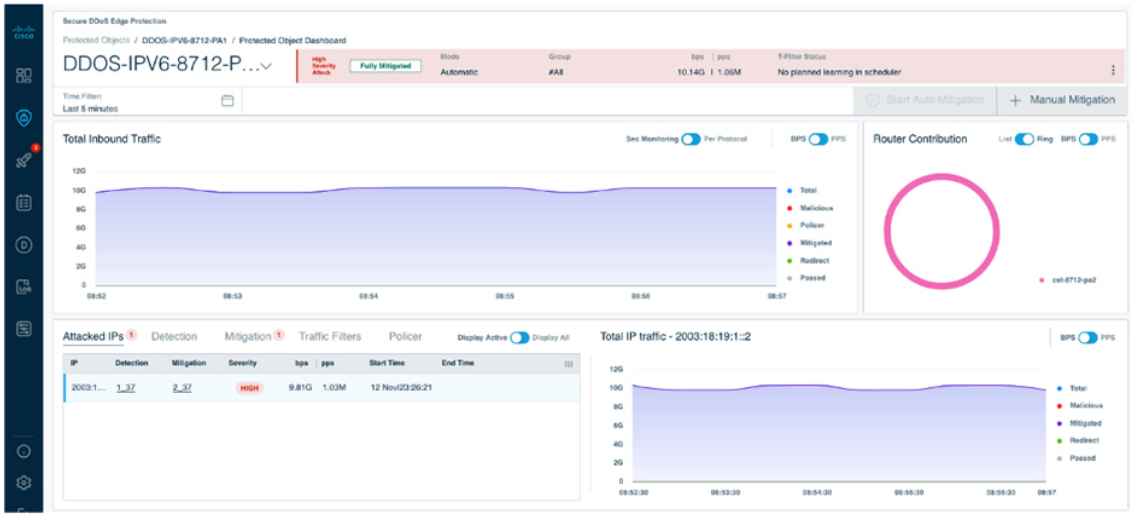
Mitigation on distributed nodes is performed using standard data plane ACLs. The ACLs contain granular information to mitigate only the attacks, and on platforms supporting User Defined Fields (UDF) can mitigate traffic by pattern matching components in the IP header. Active mitigations are shown both on the main launch screen for the Controller as well as under specific detectors.

Figure 9: Active mitigations



524699

Figure 10: Protected object mitigation details



For details on Edge DDoS Protection, see the Quick Start Guide for Edge DDoS Protection.

Example of Edge Protect deployed ACL

This is an example of an automated ACL used to mitigate an attack. In this case, the protected objects cover the 203.0.113.100 and 198.51.100.2 IP addresses. The ACL is as granular as possible to mitigate attacks in a targeted manner. ACL sequence number 1 blocks DNS packets with a specific length 228 and sequence 6 blocks an application layer attack against http.

```
RP/0/RP0/CPU0:cst-8712-pa2#show run ipv4 access-list myACL
ipv4 access-list myACL
 1 deny udp any eq 3000 host 203.0.113.100 eq domain packet-length eq 228
 6 deny tcp any eq 3400 host 198.51.100.2 eq www match-all -established -fin -psh +syn -urg
 packet-length eq 1178
 1301 permit ipv4 any any
```

Cisco Routed PON deployment

Components of Cisco Routed PON

The Cisco Routed PON solution is based on Cisco uOLT PON SFP, and the Routed PON Management applications.

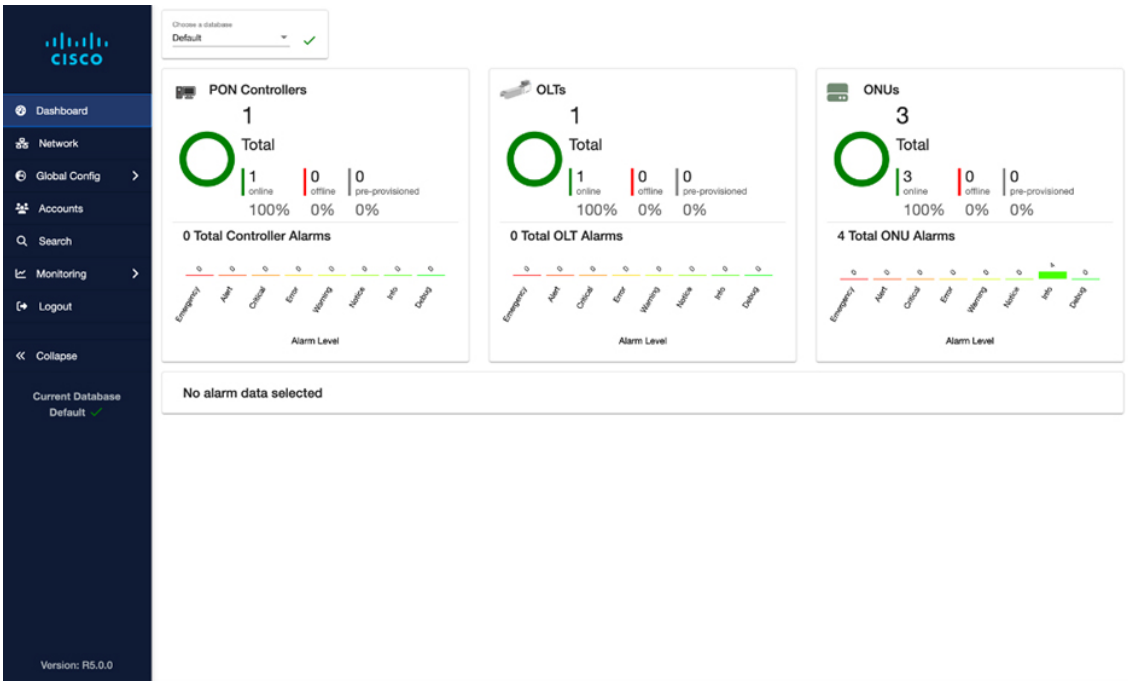
These are the major elements in manageability solution:

- Cisco Routed PON Manager
- Cisco Routed PON Controller

Cisco Routed PON Manager

Cisco Routed PON Manager is a single-page web application and an accompanying REST API that provides a graphical user interface for managing the Cisco Routed PON Network.

Figure 11: Cisco Routed PON Manager



524701

The key features of Cisco Routed PON manager include

- alarm management
- dashboard view with a summary of PON network conditions
- device monitoring and statistics
- device provisioning and management
- logging for diagnostics and troubleshooting
- PON Controller database management
- PON Manager user management
- Graphical ONU Management and Control Interface (OMCI) (and future 10G EPON OAM) service configuration tool, and
- service configuration, including VLANs, SLAs, 802.1X authentication, and DHCP Relay.

Cisco Routed PON Controller

Cisco Routed PON Controller is a stateless software that primarily act as intelligent relay to push or pull information from OLT micro plug or ONUs and transfer them to or from data store. The PON Controller is hosted as a third-party application container in router where the PON SPFs are hosted. The pseudo driver functions implemented in PON Controller encode and encapsulate requests in IEEE 1904.2 packets (L2) to

communicate with downstream devices. The Cisco PON Controller runs as a third-party application the Cisco router hosting Cisco PON pluggable OLT.

Operational data including device state, statistics, alarms, and logging, is collected and flows upstream through the management network and presented in Cisco Routed PON Manager.

Hardware support for Cisco Routed PON in Metro solution

The table lists the supported hardware for Cisco Routed PON in Metro solution.

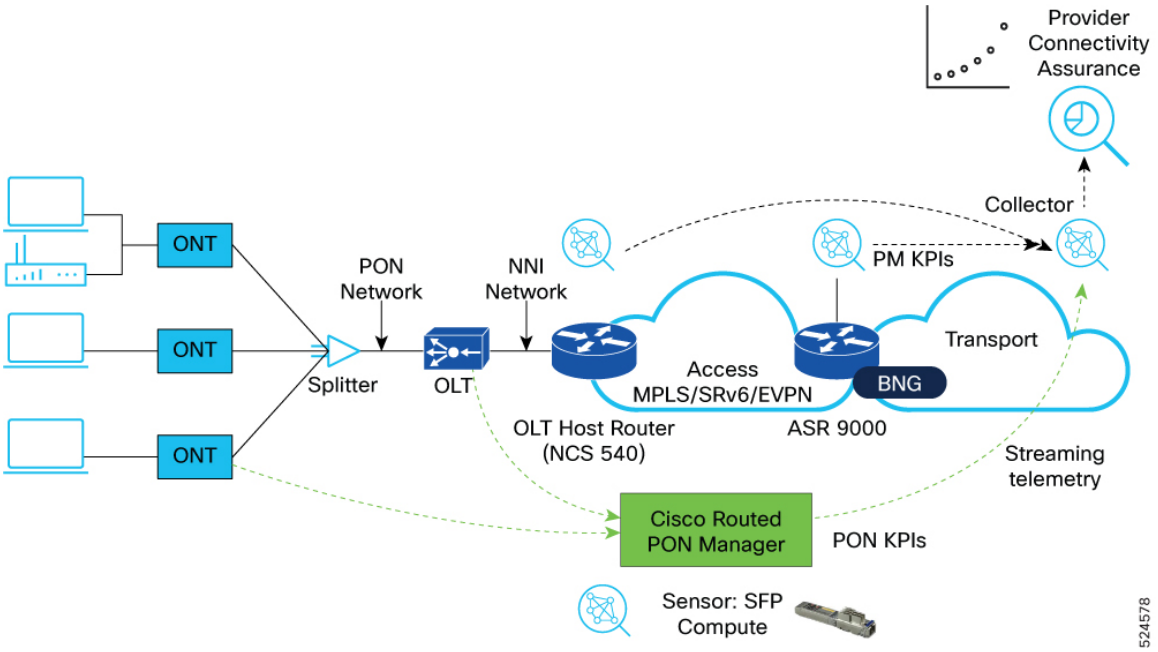
Table 1: Hardware support Matrix for Cisco Routed PON in Metro solution

Metro release	Product Id
Release 1.0	N540-24Z8Q2C-SYS
	N540-ACC-SYS
	N540-24Q8L2DD-SYS
	N540X-16Z4G8Q2C-D/A
	N540-28Z4C-SYS-D/A
	NCS-55A2-MOD-S
	NCS-57C1-48Q6D
	NCS-55A1-24Q6H-SS

Routed PON service assurance using Provider Connectivity Assurance

Provider Connectivity Assurance (PCA) provides assurance for Routed PON by combining data from the PON network, network fabric, and BNG subscriber data to create the end-to-end assurance view for PON attached endpoints.

Figure 12: Routed PON PCA integration

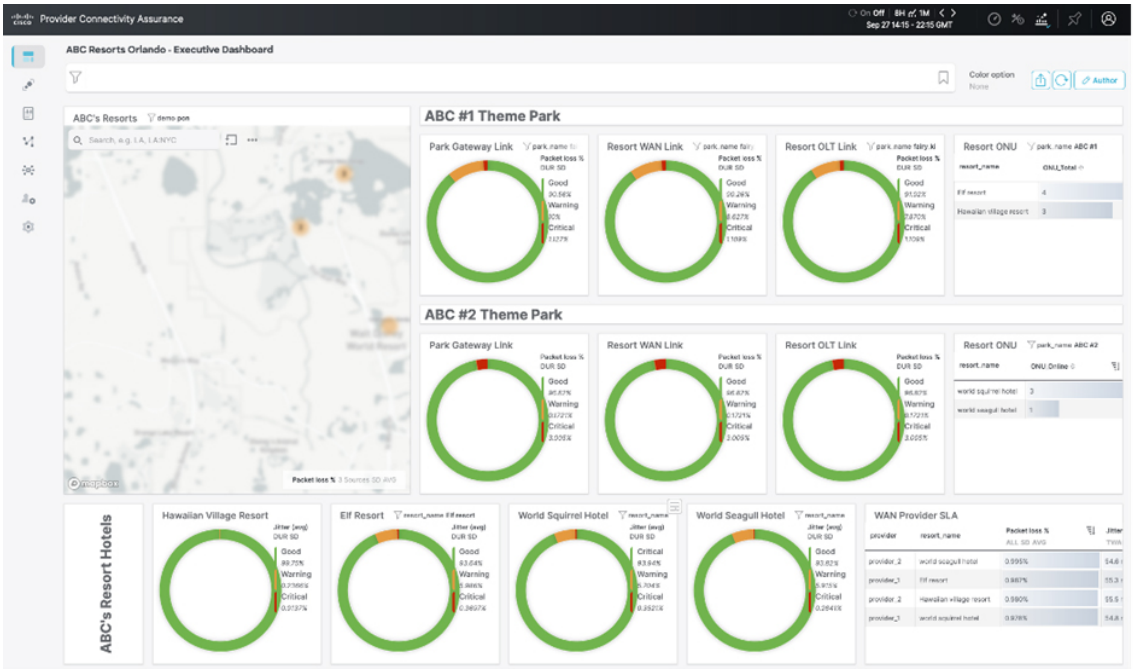


These are the PCA use cases specific to RPON service assurance:

- Integration with Cisco PON Manager to collect KPI data for the PON network
- Monitor network infrastructure data and correlate that with the PON network data. One use case is to monitor latency between PON endpoints and cnBNG user plane endpoints, correlating specific subscribers connected to a specific BNG user plane
- Integration with Cisco cnBNG user planes and control plane to provide additional per-subscriber assurance for the solution by combining BNG health and subscriber monitoring into the PCA solution

The figure depicts the Routed PON assurance dashboard.

Figure 13: Routed PON assurance dashboard



524702