Nokia Validated Design

Collapsed Spine EVPN VXLAN

3HE-21913-AAAA-TQZZA
Issue 1
July 2025

Nokia is committed to diversity and inclusion. We are continuously reviewing our customer documentation and consulting with standards bodies to ensure that terminology is inclusive and aligned with the industry. Our future customer documentation will be updated accordingly.

Contents

List of Tables

List of Figures

3HE-21913-AAAA-TQZZA
July 2025
Issue 1

1 Executive summary

Nokia Validated Designs (NVDs) is a workstream dedicated to producing validated recommendations to the consumer about Nokia's portfolio across market segments.

NVDs are accomplished with extensive requirement analysis from a multitude of customers along with deep research of the technology development in the industry segment to form the solution's design.

After the design has been compiled, it goes through an intense array of hardware, software, traffic, and failure tests to form the validated design. The resultant design and collateral provide the consumer with a template that can be used to deploy the solution in their own environment.

NVDs are structured as core and ancillary (extension) designs. This document outlines a collapsed spine EVPN VXLAN design, describing various physical and logical connectivity aspects and associated technologies involved in a single site, single-tiered data center architecture with EVPN as the control plane and VXLAN as the data plane.

2 Reference architecture overview

2.1 Design considerations and components

A high-level overview of the collapsed spine topology is shown in Figure 1.

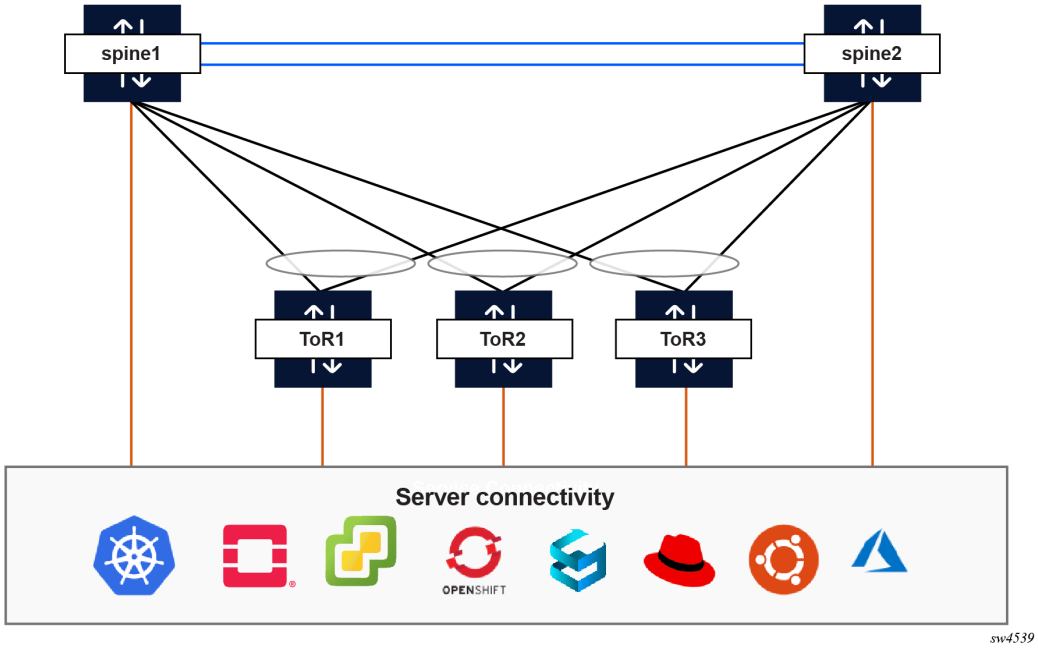

Figure 1 **Collapsed spine EVPN VXLAN NVD architecture**

This section describes the various components involved in this validated design and the design and technology choices that were made. In general, a collapsed spine design provides the following advantages:

- reduction of the required number of devices and capacity of the devices in data centers where the scale-out and size of the data center are known to be limited and the main considerations become cost and power utilization
- re-use of legacy Layer 2 switches (even Layer 2 switches of other vendors) as top of rack (ToR) switches in more modern data center designs

  A collapsed spine design allows you to connect legacy switches or switches with only a Layer 2 license (thus minimizing investment in network infrastructure) while still moving into a modern architecture, which allows room to grow into a scaled-out 3-stage Clos design when the need arises.

The collapsed spine architecture has the following design considerations:

- The design is a single-tier design where the spines and leafs of a 3-stage Clos fabric are collapsed into a single layer (tier), referred to as the collapsed spine (core), with various IPv4 and IPv6 server attachments.
- The two spines are connected using point-to-point Layer 3 interfaces. These interfaces are enabled with IPv6, which automatically generates IPv6 link-local addresses per-interface. Peers are discovered using IPv6 neighbor discovery (ND).
- A single BGP (MP-BGP) session is dynamically established using these IPv6 link-local addresses, carrying multiple AFIs/SAFIs (IPv4, IPv6, EVPN) as needed.
- During the establishment of this session, the extended next-hop encoding capability is exchanged, enabling IPv4 routes to be advertised with IPv6 next hops (as described in RFC 8950). This enables the point-to-point connectivity between the spines to be IPv6 link-local only, allowing operators to move away from the operational overhead of IPv4 underlay management while still providing an IPv4 overlay.
- All VXLAN-specific constructs are configured on the spines, making them the VXLAN tunnel endpoints (VTEPs) in this fabric. Because these spines are VTEPs, Nokia's platforms based on Broadcom's Trident chipset are positioned at this layer.
- The ToR switches are multihomed to the spines using all-active Ethernet segments (ESs) and act as pure Layer 2 switches in this design.
- In such a design, servers can be connected anywhere – either directly to the spines (single-homed or multihomed) or to the ToR switches.
- This design covers the following server connectivity options, across both IPv4 and IPv6 based endpoints:

- Layer 2 untagged interfaces
- Layer 2 tagged interfaces
- Layer 3 point-to-point interfaces (exported as EVPN Type-5 routes into the fabric)
- two-way ES-based Link Aggregation Group (LAG) in all-active multihoming mode
- two-way ES-based LAG in single-active multihoming mode

2.2 High-level operational workflow

Figure 2 shows a high-level operational workflow for the NVD-based fabric deployment and lifecycle management. The intent-based approach, combined with the prescriptive nature of the validated design and the flexibility of Nokia's Event Driven Automation (EDA), makes the deployment of the fabric effortless and reliable.

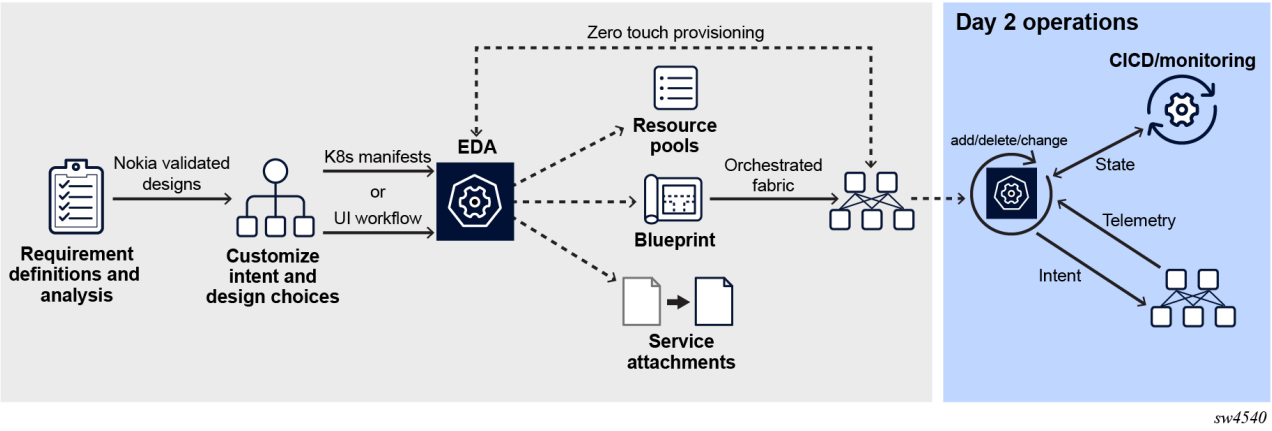

Figure 2 **High-level operational flow diagram**

- As with traditional deployments, broad customer requirements are gathered based on the applications and workloads that are going to be operational in the data center.
- After customer requirements are analyzed and collated into infrastructure requirements, they are converted to an intent by customizing the closest available NVD (in this case, the collapsed spine EVPN VXLAN NVD).
- After the customization is complete, the intent can be orchestrated onto EDA using Kubernetes manifest files or the EDA UI.
- EDA then creates the necessary blueprints and resource pools. The generated configuration is pushed to the fabric nodes that have been onboarded using zero-touch provisioning (ZTP).
- After the fabric is deployed, EDA provides comprehensive telemetry options that can be connected to CI/CD pipelines to modify the intent and the fabric as needed.

- Because EDA as a platform does not need to be reinstalled for new patches or apps, it provides a high degree of flexibility and customizability for modern DC fabric needs.

3 Network deployment

3.1 High-level design

Figure 3 shows a high-level design of the fabric. The topology is a collapsed spine fabric with BGP EVPN as the control plane and VXLAN as the data plane encapsulation method with point-to-point layer 3 links between the two spines. These point-to-point interfaces are configured with IPv6-link local addressing (as shown in Figure 3), with each spine advertising its IPv4 loopback address with an IPv6 next-hop (as described in RFC 8950), as shown in Figure 4. Each node in Figure 4 is labeled with sample IPv4 addresses assigned to the loopback interface.

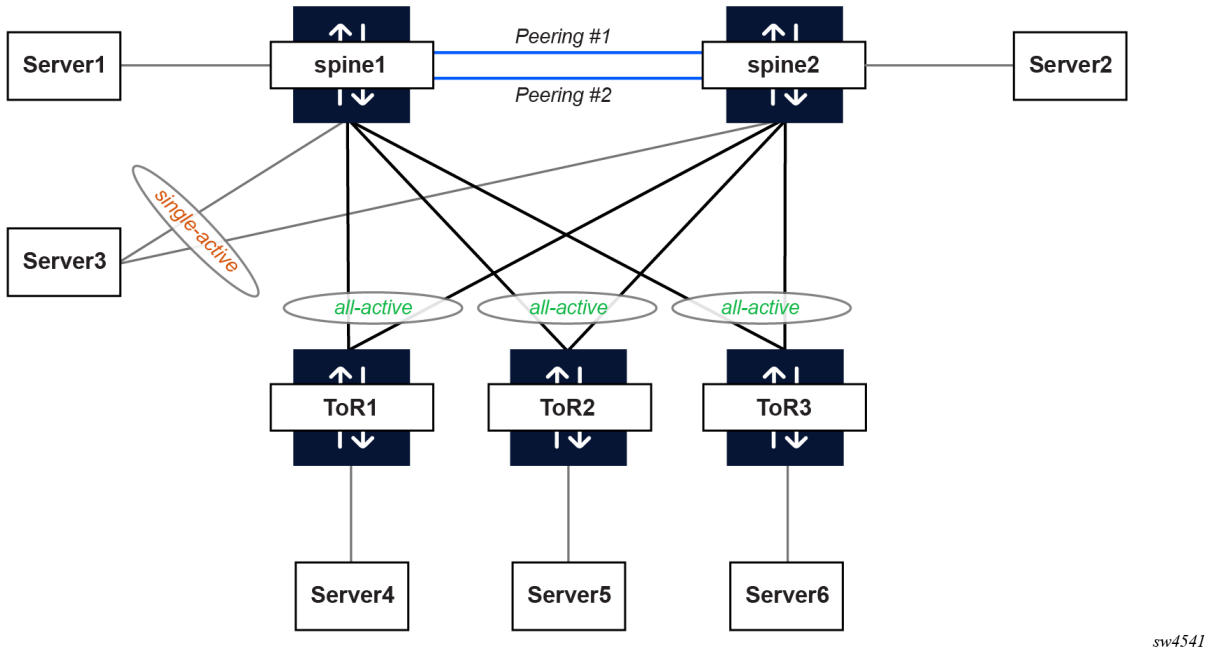

Figure 3 **High-level diagram with underlay and overlay**

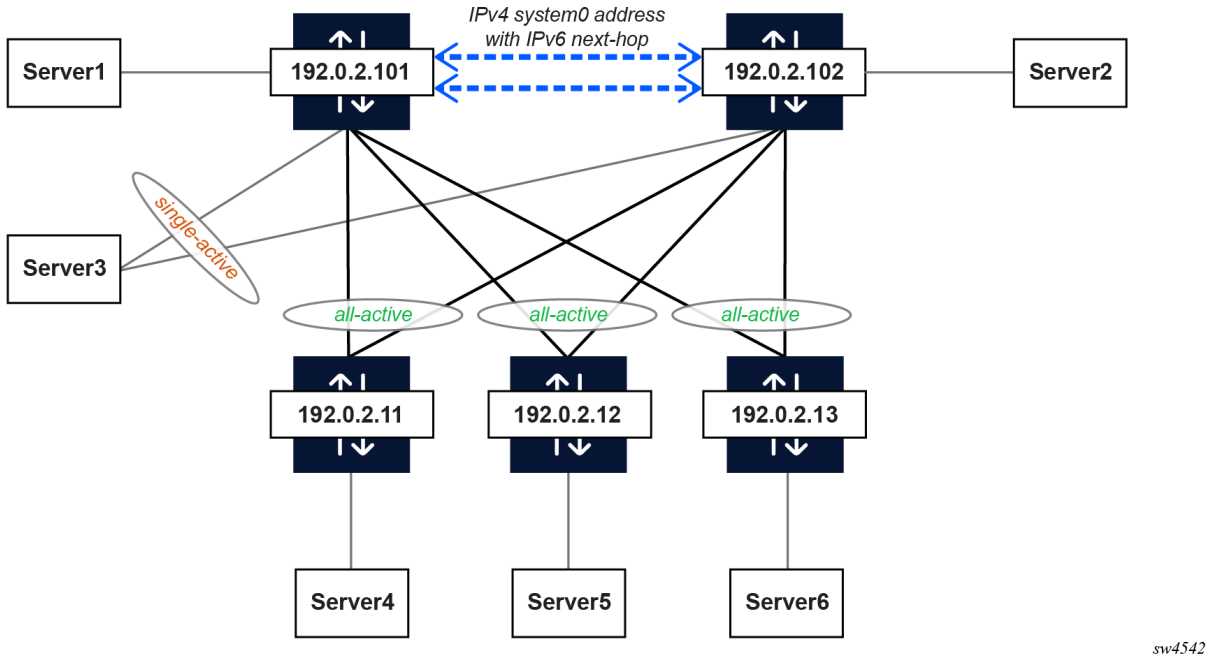

Figure 4 **IPv4 system0 addresses advertised with an IPv6 next-hop**

Integrated routing and bridging (IRB) interfaces are configured on the spines using a distributed anycast gateway model. These IRB interfaces act as the default gateway for all servers in the fabric. Servers can connect either directly to the spines or to the ToR switches, depending on customer requirement.

For routing between VNIs, this design uses an asymmetric routing model (as described in RFC 9135), along with symmetric routing using EVPN Type-5 routes for certain subnets (servers connected via Layer 3 point-to-point interfaces).

3.2 Platform positioning

This section describes the Nokia platforms positioned for different roles in the collapsed spine EVPN VXLAN validated design. Figure 5 provides a visual depiction while Table 1 lists all platforms and their count in the fabric.

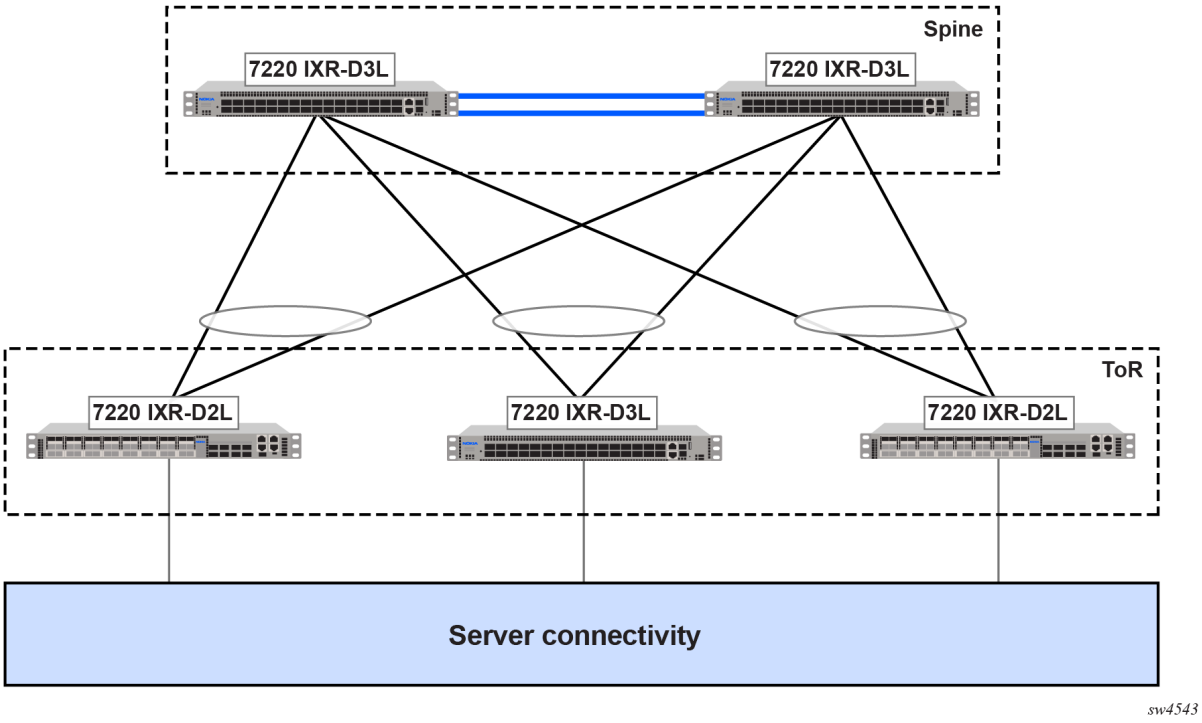

Figure 5 **High-level platform positioning**

Table 1 **Platform positioning**

Device	Role	Count
7220-IXR-D3L	Collapsed spine	2
7220-IXR-D2L	Leaf/ToR	2
7220-IXR-D3L	Leaf/ToR	1

Note: Alternate platforms can be positioned in the roles shown above based on cost, hardware, and performance requirements.

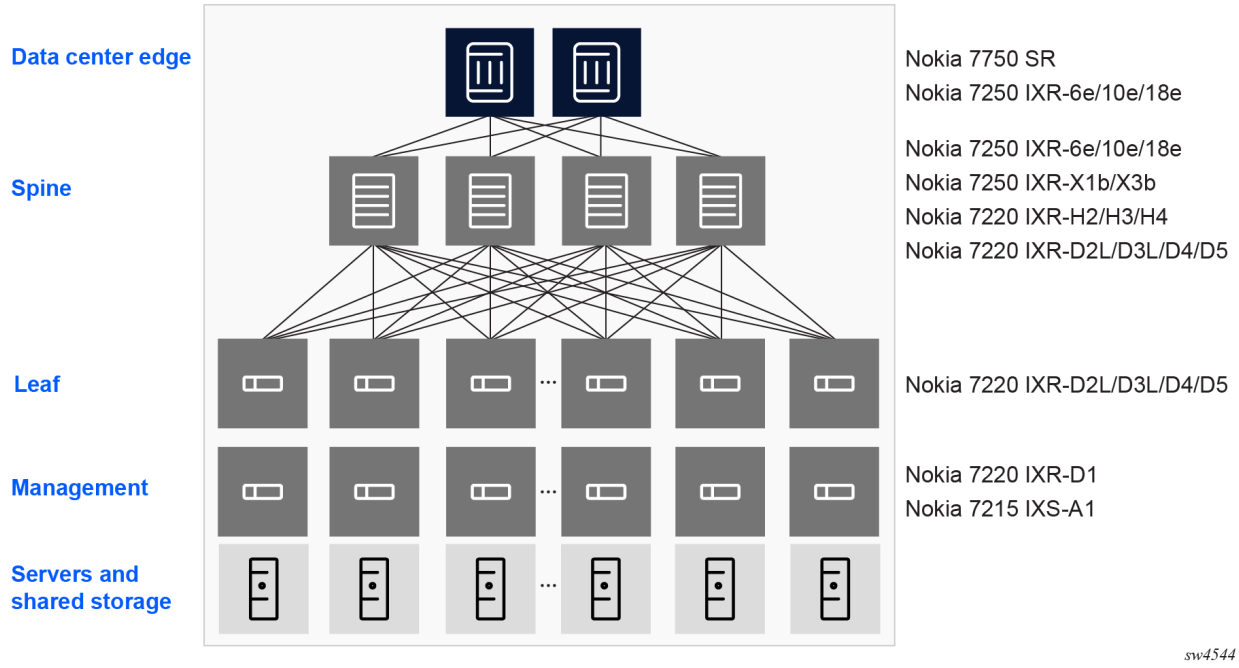

Figure 6 **Nokia data center portfolio**

3.3 Traffic patterns

In this section, we describe common traffic patterns that are validated in the collapsed spine EVPN VXLAN NVD.

These traffic patterns include forwarding across Layer 2 tagged and untagged interfaces (single-homed and multihomed), Layer 3 interfaces, two-way all-active Ethernet segment LAG, and two-way single-active Ethernet segment LAG.

Figure 7 and Figure 8 show traffic ingress on a single-homed interface and egress out of an Ethernet segment member interface (either local or remote).

- In Figure 7, when there is an interface local to the ingress leaf (VTEP) that is also part of the egress Ethernet segment, forwarding occurs using the rules of local bias where the local egress member of the Ethernet segment is selected as the exit interface.
- In Figure 8, if the local egress member link of the Ethernet segment is not available (down), packets are forwarded over the fabric by encapsulating with VXLAN headers towards a remote leaf (VTEP) that is also part of the same Ethernet segment, eventually leaving via the member interface of this Ethernet segment.

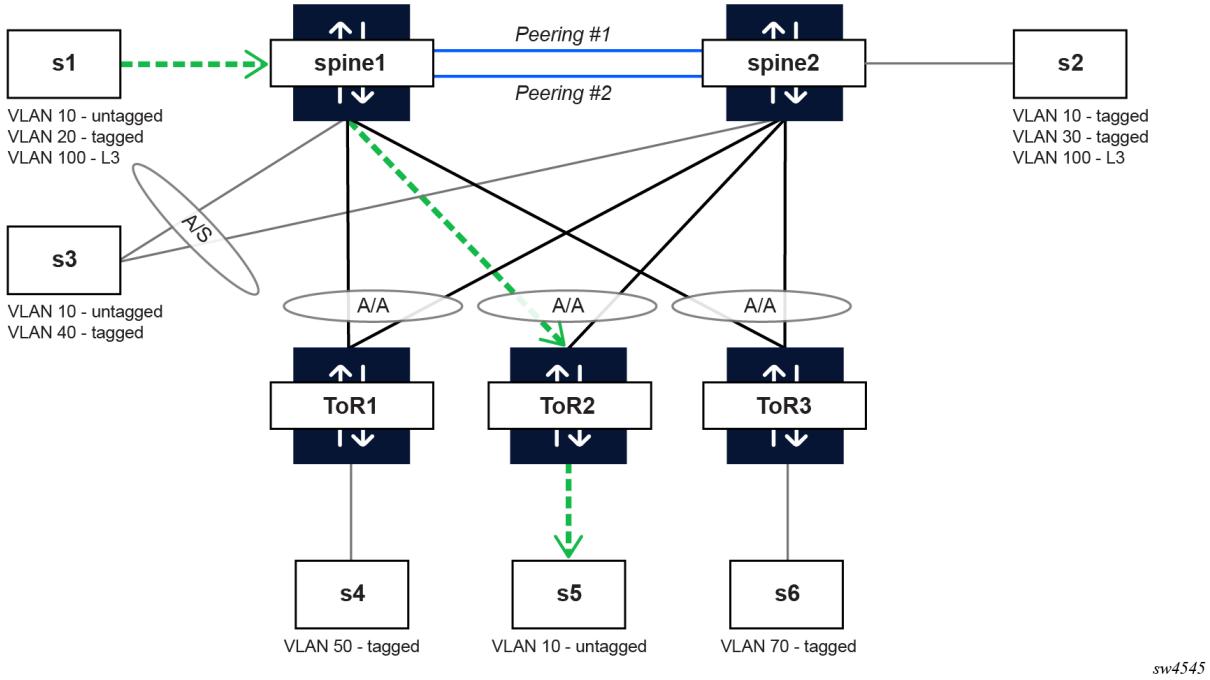

Figure 7 **Packet flow for Layer 2 tagged and untagged traffic using local bias forwarding**

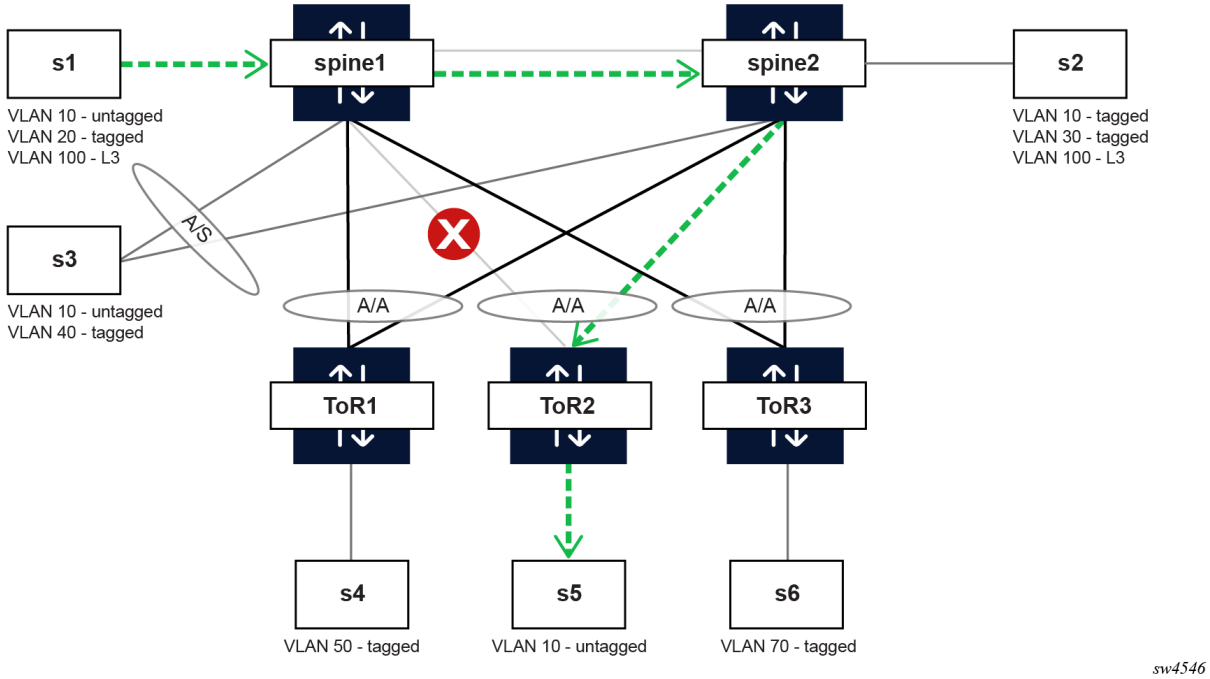

Figure 8 **Packet flow for Layer 2 tagged and untagged traffic exiting via a remote VTEP when local member interface of Ethernet segment is down on ingress VTEP**

Figure 9 demonstrates traffic between single-homed tagged or untagged interfaces across both spines (with VXLAN encapsulation). While traffic is shown using one path only, equal cost paths between the spines are used based on variability in traffic patterns.

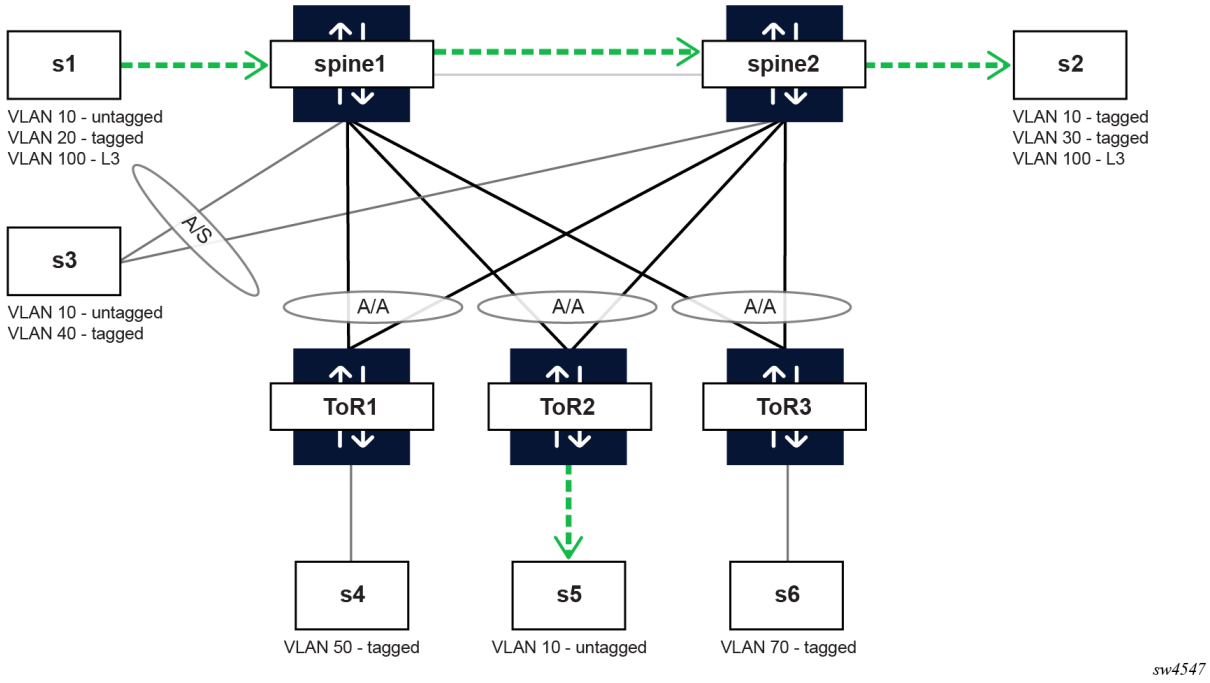

Figure 9 **Packet flow between tagged or untagged interfaces across spines**

Figure 10 and Figure 11 demonstrate the traffic patterns for a destination that is behind a single-active Ethernet segment.

- Figure 10 demonstrates traffic ingress via the active VTEP of a single-active Ethernet segment and uses local bias forwarding rules to send out another locally attached Ethernet segment.
- Figure 11 demonstrates traffic ingress on the remote spine via a single-homed interface. Traffic is forwarded over the fabric by encapsulating VXLAN headers and then sent out the interface of the single-active Ethernet segment of the active, remote VTEP. On the ingress VTEP, the Ethernet segment resolves to the VTEP address of the active node only.

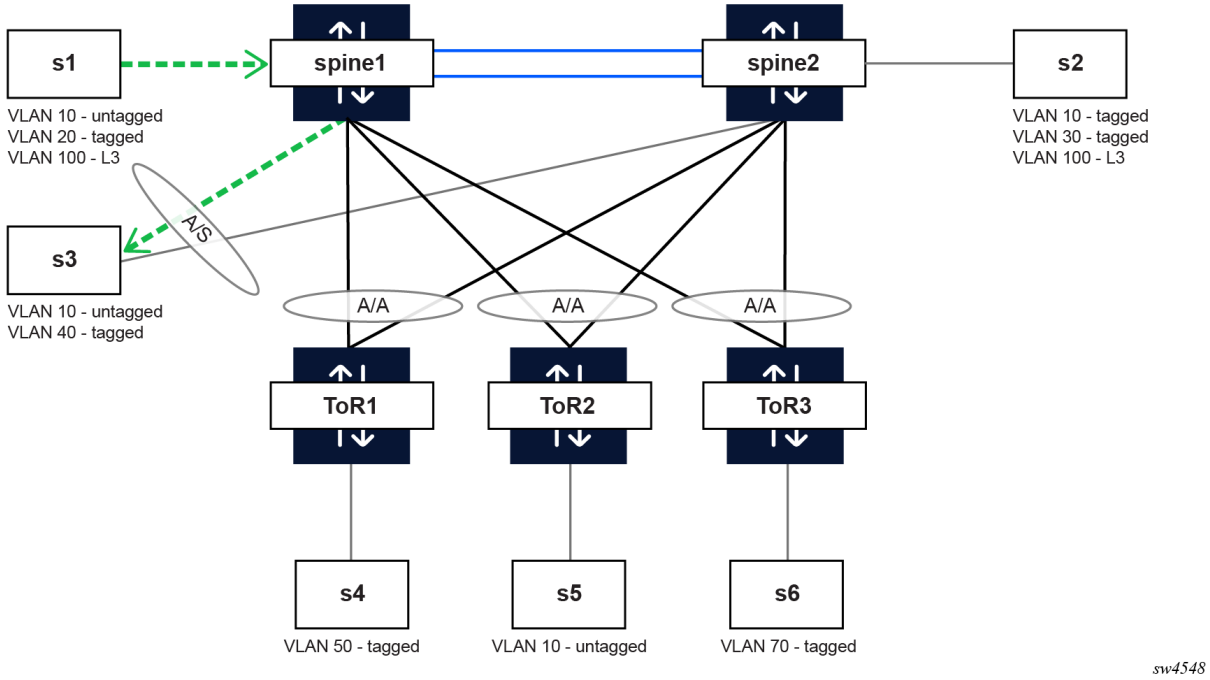

Figure 10 **Single-active Ethernet segment with local bias forwarding**

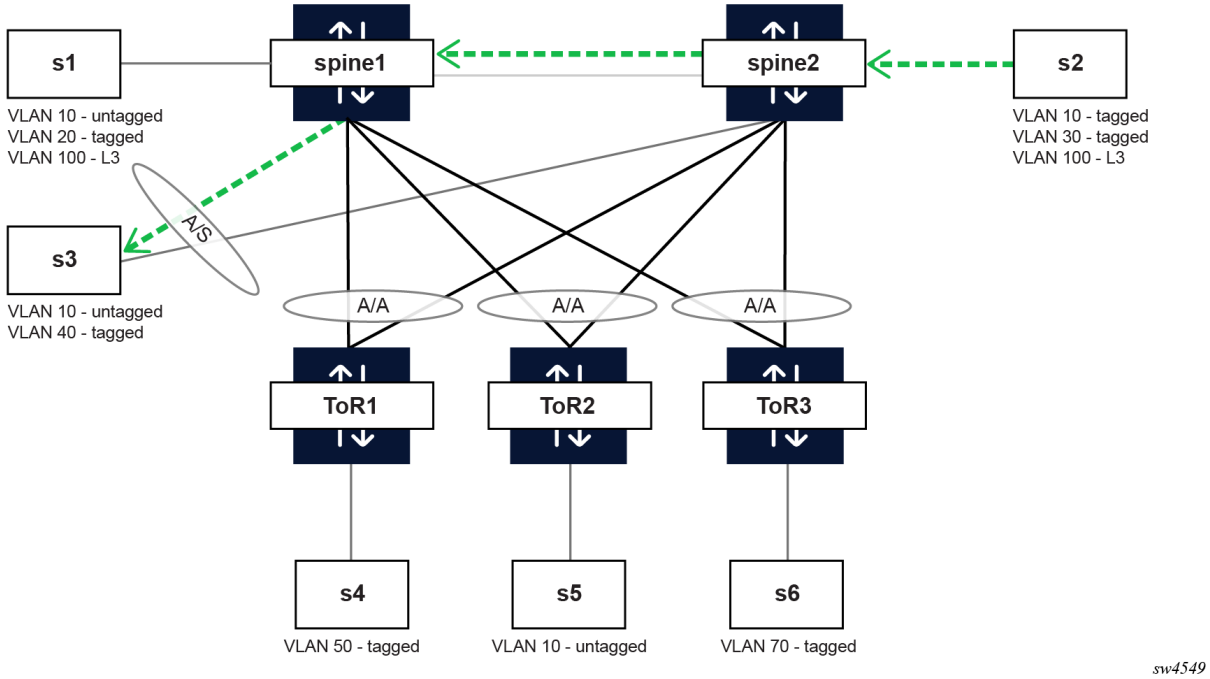

Figure 11 **Single-active Ethernet segment with forwarding over fabric**

Figure 12 demonstrates the traffic pattern for sources and destinations behind ToR switches.

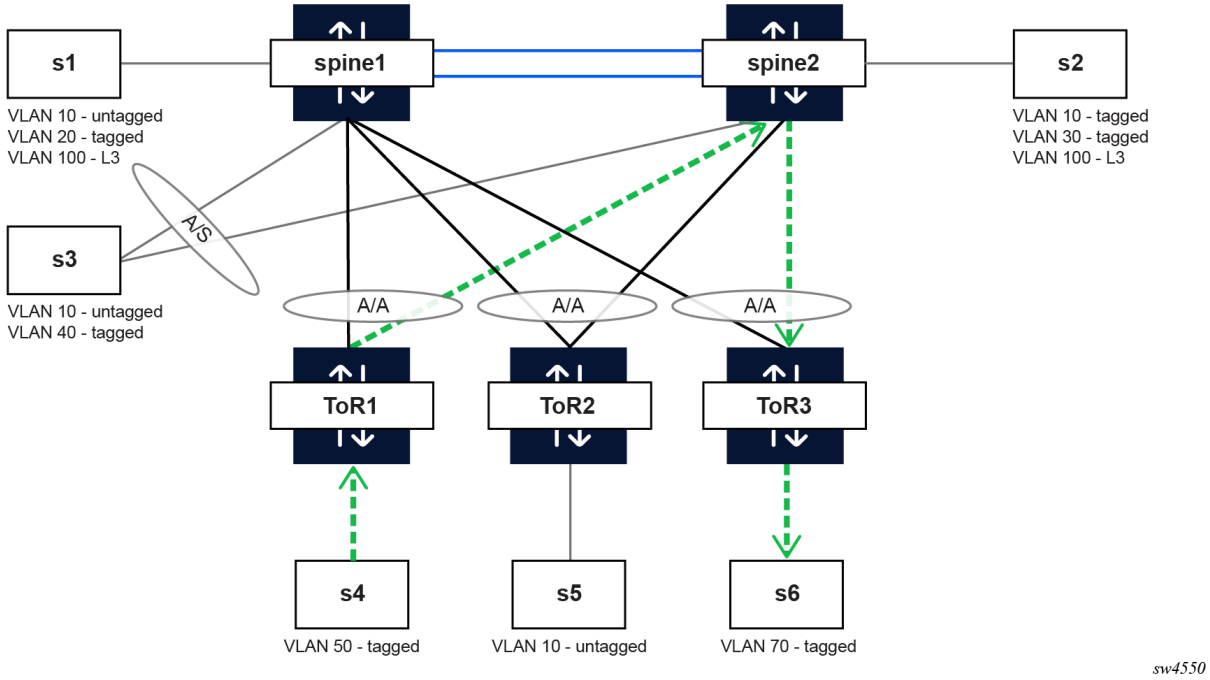

Figure 12 **Traffic flow for source and destination behind ToR switches**

Figure 13 and Figure 14 demonstrate traffic patterns for a source behind a ToR switch communicating with a destination behind the single-active Ethernet segment, depending on where the ToR hashes the flow (either to spine1 or spine2).

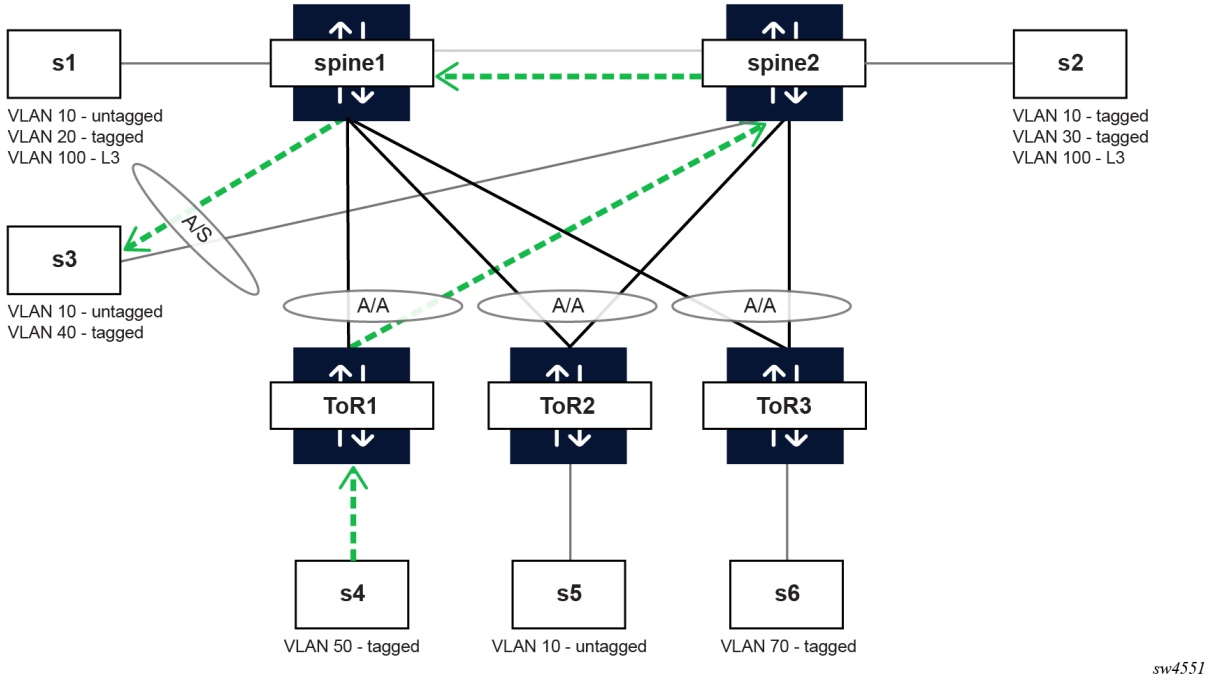

Figure 13 **Traffic flow for source behind ToR communicating with a destination behind single-active Ethernet segment when packet is hashed to non-DF**

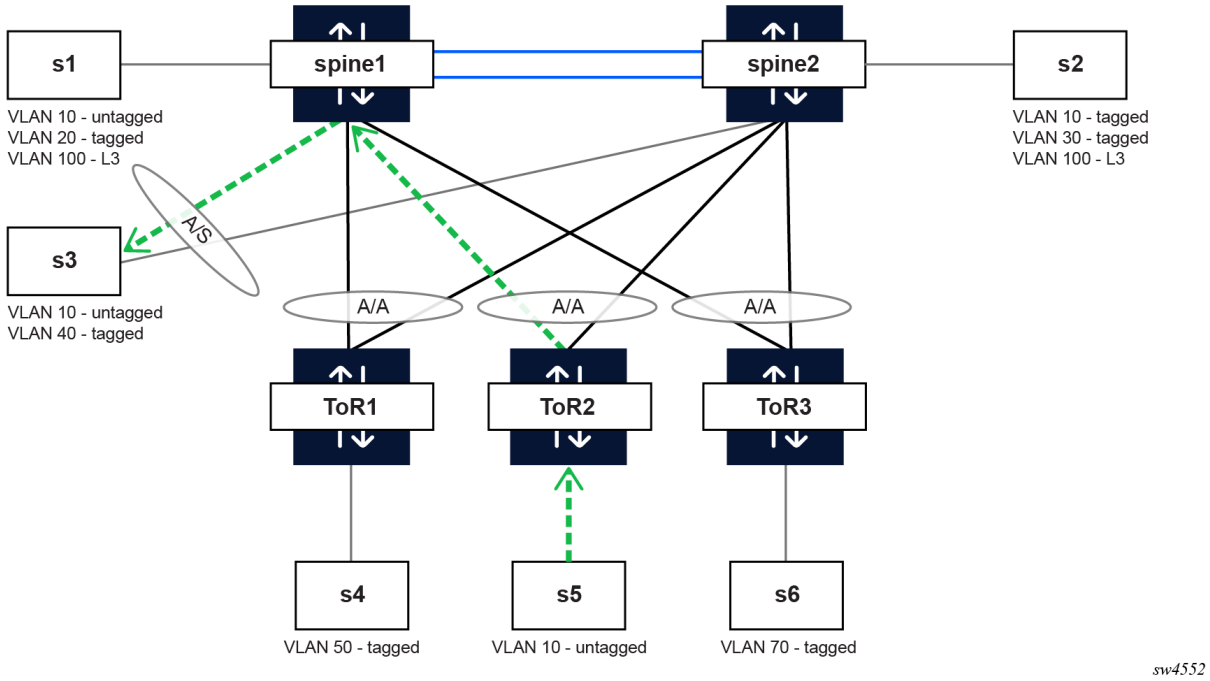

Figure 14 **Traffic flow for source behind ToR communicating with a destination behind single-active Ethernet segment when packet is hashed to DF**

4 Feature configuration

4.1 Underlay with IPv6 link-local addressing for point-to-point interfaces between the collapsed spines

The point-to-point interfaces between the collapsed spines are enabled for IPv6 only, with link-local addressing. IPv6 ND is used to resolve the peers' address. The addressing is enabled on subinterfaces within each physical interface. These subinterfaces are then mapped to the default network-instance.

The system0 interface, used as the VTEP address, is configured with a /32 address. These addresses are used as the source and destination addresses in the outer IP header for VXLAN tunnels. In this document, the IPv4 documentation range 192.0.2.0/24 is used for assignment.

```
// point-to-point links to spine2

A:admin@d3l-29-spine1# info interface ethernet-1/{31,32}
    interface ethernet-1/31 {
        admin-state enable
        subinterface 0 {
            admin-state enable
            ipv6 {
                admin-state enable
                router-advertisement {
                    router-role {
                        admin-state enable
                        max-advertisement-interval 10
                        min-advertisement-interval 4
                    }
                }
            }
        }
    }
    interface ethernet-1/32 {
        admin-state enable
        subinterface 0 {
            admin-state enable
            ipv6 {
                admin-state enable
                router-advertisement {
                    router-role {
                        admin-state enable
                        max-advertisement-interval 10
                        min-advertisement-interval 4
                    }
                }
            }
        }
    }

// system0 configuration

A:admin@d3l-29-spine1# info interface system0
    subinterface 0 {
```

```
        admin-state enable
        ipv4 {
            admin-state enable
            address 192.0.2.2/32 {
            }
        }
    }
```

Example 1 **Configuration of point-to-point interfaces and system0 interface**

4.2 Default network-instance

The point-to-point interfaces between the collapsed spines are mapped to the default network-instance in SR Linux. Additionally, the system0 subinterface used as the VTEP source address is also mapped to the default network-instance. Because the NVD uses IPv4 addressing for the system0 interface (which is used for VXLAN tunnels), the IPv6 forwarding check must be disabled as IPv4 packets received on an IPv6-only interface are dropped by default. This is achieved by setting the *ip-forwarding receive-ipv4-check* configuration option to *false*.

```
// configuration of default network-instance

A:admin@d31-29-spine1# info network-instance default
    type default
    admin-state enable
    description "fabric: dc1-collapsed-spine role: leaf"
    router-id 192.0.2.2
    ip-forwarding {
        receive-ipv4-check false
    }
    interface ethernet-1/31.0 {
    }
    interface ethernet-1/32.0 {
    }
    interface system0.0 {
    }

*snip*
```

Example 2 **Configuration snippet of the default network-instance**

4.3 BGP for underlay and overlay routes

The NVD uses an eBGP design with each collapsed spine assigned a unique BGP ASN, utilizing MP-BGP functionality with multiple address families advertised as capabilities over a single BGP session. The eBGP sessions are configured for dynamic discovery, leveraging the IPv6 link-local underlay design and IPv6 ND capabilities. In addition, the following also apply to BGP:

- EDA-generated routing policies for advertising underlay IPv4 system0 routes and IPv4/IPv6 overlay EVPN routes
- configuration option to allow IPv4 routes to be advertised with IPv6 next hops

- configuration option to accept receipt of IPv4 routes with IPv6 next hops
- multipath enabled for IPv4 unicast, IPv6 unicast, and L2VPN EVPN AFIs/SAFIs
- configuration option to enable rapid withdrawal of BGP routes and rapid update of EVPN routes

```
// BGP configuration on spine1

A:admin@d3l-29-spine1# info network-instance default protocols bgp
    admin-state enable
    autonomous-system 65501
    router-id 192.0.2.2
    dynamic-neighbors {
        interface ethernet-1/31.0 {
            peer-group bgpgroup-ebgp-dc1-collapsed-spine
            allowed-peer-as [
                65502
            ]
        }
        interface ethernet-1/32.0 {
            peer-group bgpgroup-ebgp-dc1-collapsed-spine
            allowed-peer-as [
                65502
            ]
        }
    }
    ebgp-default-policy {
        import-reject-all true
        export-reject-all true
    }
    afi-safi evpn {
        admin-state enable
        multipath {
            allow-multiple-as true
            ebgp {
                maximum-paths 64
            }
            ibgp {
                maximum-paths 64
            }
        }
        evpn {
            inter-as-vpn true
            rapid-update true
        }
    }
    afi-safi ipv4-unicast {
        admin-state enable
        multipath {
            allow-multiple-as true
            ebgp {
                maximum-paths 2
            }
            ibgp {
                maximum-paths 2
            }
        }
        ipv4-unicast {
            advertise-ipv6-next-hops true
```

```
            receive-ipv6-next-hops true
        }
        evpn {
            rapid-update true
        }
    }
    afi-safi ipv6-unicast {
        admin-state enable
        multipath {
            allow-multiple-as true
            ebgp {
                maximum-paths 2
            }
            ibgp {
                maximum-paths 2
            }
        }
        evpn {
            rapid-update true
        }
    }
    preference {
        ebgp 170
        ibgp 170
    }
    route-advertisement {
        rapid-withdrawal true
        wait-for-fib-install false
    }
    group bgpgroup-ebgp-dc1-collapsed-spine {
        admin-state enable
        export-policy [
            ebgp-isl-export-policy-dc1-collapsed-spine
        ]
        import-policy [
            ebgp-isl-import-policy-dc1-collapsed-spine
        ]
        failure-detection {
            enable-bfd true
            fast-failover true
        }
        afi-safi evpn {
            admin-state enable
        }
        afi-safi ipv4-unicast {
            admin-state enable
            ipv4-unicast {
                advertise-ipv6-next-hops true
                receive-ipv6-next-hops true
            }
        }
        afi-safi ipv6-unicast {
            admin-state enable
        }
    }
```

Example 3 **BGP configuration on spine1**

4.4 MTU

System-wide maximum transmission units (MTUs) are configured globally to accommodate larger-sized packets (considering 50 Bytes overhead is added as part of the overall VXLAN encapsulation).

```
// system-wide default MTU configuration on collapsed spines and ToR switches

A:admin@d3l-29-spine1# info system mtu
    default-port-mtu 9412
    default-l2-mtu 9412
    default-ip-mtu 9200
```

Example 4 **Configuration of system-wide default MTUs on a Nokia 7220 IXR-D3L**

4.5 BFD

Bidirectional forwarding detection (BFD) is enabled on the links between the collapsed spines. BGP is enabled for fast failover using BFD (with a failure detection time of 750 ms).

```
// BGP configuration on point-to-point subinterface

A:admin@d3l-29-spine1# info bfd
    subinterface ethernet-1/31.0 {
        admin-state enable
        desired-minimum-transmit-interval 250000
        required-minimum-receive 250000
        detection-multiplier 3
        minimum-echo-receive-interval 250000
    }
    subinterface ethernet-1/32.0 {
        admin-state enable
        desired-minimum-transmit-interval 250000
        required-minimum-receive 250000
        detection-multiplier 3
        minimum-echo-receive-interval 250000
    }

// BGP enabled for fast-failover with BFD

A:admin@d3l-29-spine1# info network-instance default protocols bgp group bgpgroup-ebgp-
dc1-collapsed-spine failure-detection
    enable-bfd true
    fast-failover true
```

Example 5 **Configuration of BFD and BGP enabled for fast failover**

4.6 LLDP

Link Layer Discovery Protocol (LLDP) is used to discover neighboring devices.

```
// LLDP enabled for neighbor discovery

A:admin@d3l-29-spine1# info system lldp
    interface ethernet-1/23 {
        admin-state enable
    }
    interface ethernet-1/25 {
        admin-state enable
    }
    interface ethernet-1/26 {
        admin-state enable
    }
    interface ethernet-1/27 {
        admin-state enable
    }
    interface ethernet-1/29 {
        admin-state enable
    }
    interface ethernet-1/31 {
        admin-state enable
    }
    interface ethernet-1/32 {
        admin-state enable
    }
```

Example 6 **LLDP configuration**

4.7 Layer 2 server-facing interfaces

Untagged and tagged Layer 2 server-facing interfaces are tested as part of this NVD. A sample configuration is provided below with multiple subinterfaces configured on an interface, one tagged and another untagged. Use of subinterfaces in this fashion allows for a logical separation of the expected traffic on the physical interface. These subinterfaces are then mapped to their respective MAC VRFs (shown later in this document).

```
// Layer 2 untagged and tagged subinterfaces

A:admin@d3l-29-spine1# info interface ethernet-1/27
    admin-state enable
    vlan-tagging true
    subinterface 20 {
        type bridged
        description tagged-v20
        admin-state enable
        vlan {
            encap {
                single-tagged {
                    vlan-id 20
                }
```

```
            }
        }
    }
    subinterface 100 {
        type routed
        description spine1-l3-1
        admin-state enable
        ip-mtu 9200
        ipv4 {
            admin-state enable
            address 172.16.100.2/31 {
                primary
            }
            arp {
                timeout 14400
            }
        }
        ipv6 {
            admin-state enable
            address 2001:db8:0:100::2/127 {
                primary
            }
        }
        vlan {
            encap {
                single-tagged {
                    vlan-id 100
                }
            }
        }
    }
    subinterface 4096 {
        type bridged
        description untagged-v10
        admin-state enable
        vlan {
            encap {
                untagged {
                }
            }
        }
    }
```

Example 7 **Configuration of Layer 2 untagged and tagger server-facing interfaces**

4.8 All-active ES-based LAG

An all-active Ethernet segment is tested as part of this NVD. Ethernet segments are supported natively within EVPN as a standard, allowing more than two VTEPs for multihoming. The configuration includes the following:

- mapping the physical interface (meant to be part of a LAG in the case of this NVD) to a LAG interface
- configuring the LAG interface with required LACP parameters
- configuring an Ethernet segment (and all required parameters) and mapping it to the respective LAG interface

- setting the designated forwarder (DF) election activation timer to 0 (the default timer is 3 seconds); this timer controls the delay of transition from non-DF to DF

```
// physical interface mapped to LAG interface

A:admin@d3l-29-spine1# info interface ethernet-1/25
    description spine1-spine2-tor1-lag
    admin-state enable
    ethernet {
        aggregate-id lag1
        lacp-port-priority 32768
        reload-delay 100
    }

// LAG interface configured with untagged/tagged subinterfaces and LACP parameters

A:admin@d3l-29-spine1# info interface lag1
    description spine1-spine2-tor1-lag
    admin-state enable
    vlan-tagging true
    subinterface 50 {
        type bridged
        description tagged-v50
        admin-state enable
        vlan {
            encap {
                single-tagged {
                    vlan-id 50
                }
            }
        }
    }
    lag {
        lag-type lacp
        min-links 1
        lacp-fallback-mode static
        lacp-fallback-timeout 60
        lacp {
            interval FAST
            lacp-mode ACTIVE
            admin-key 5
            system-id-mac 00:00:00:00:00:12
            system-priority 32768
        }
    }

// Ethernet segment configuration for all-active multihoming mode

A:admin@d3l-29-spine1# info system network-instance protocols
    evpn {
        ethernet-segments {
            bgp-instance 1 {
                ethernet-segment spine1-spine2-tor1-lag {
                    admin-state enable
                    esi 00:00:00:00:00:00:12:00:00:00
                    multi-homing-mode all-active
                    interface lag1 {
                    }
                    df-election {
```

```
                        timers {
                            activation-timer 0
                        }
                        algorithm {
                            type default
                        }
                    }
                }
                ethernet-segment spine1-spine2-tor2-lag {
                    admin-state enable
                    esi 00:00:00:00:00:12:00:00:00:00
                    multi-homing-mode all-active
                    interface lag2 {
                    }
                    df-election {
                        timers {
                            activation-timer 0
                        }
                        algorithm {
                            type default
                        }
                    }
                }
                ethernet-segment spine1-spine2-tor3-lag {
                    admin-state enable
                    esi 00:00:00:00:12:00:00:00:00:00
                    multi-homing-mode all-active
                    interface lag3 {
                    }
                    df-election {
                        timers {
                            activation-timer 0
                        }
                        algorithm {
                            type default
                        }
                    }
                }
                ethernet-segment spine1-spine2-tor4-lag {
                    admin-state enable
                    esi 00:00:00:12:00:00:00:00:00:00
                    multi-homing-mode single-active
                    interface lag4 {
                    }
                    df-election {
                        timers {
                            activation-timer 0
                        }
                        interface-standby-signaling-on-non-df {
                        }
                        algorithm {
                            type preference
                            preference-alg {
                                preference-value 800
                                capabilities {
                                    non-revertive true
                                }
                            }
                        }
                    }
                }
            }
        }
```

```
    }
    bgp-vpn {
        bgp-instance 1 {
        }
    }
```

Example 8 **Configuration of all-active ES-based LAG**

4.9 Single-active ES-based LAG

Single-active Ethernet segments (with port-active functionality, described in IETF draft https://www.ietf.org/archive/id/draft-ietf-bess-evpn-mh-pa-10.html, as of March 2025) are tested as part of this NVD. This design is useful if the server requires only a single link to be active for proper functioning, while still offering server-uplink redundancy if the active link goes down.

During steady state, only the active link forwards traffic in such a design. This is enforced by sending LACP *out-of-sync* PDUs over the member interface of the LAG on the non-DF VTEP. SR Linux also supports powering off the port (by shutting off the laser) in cases where the server does not support LACP.

If the active link goes down, the directly connected VTEP (which was the DF for that Ethernet segment) withdraws its EVPN Type-4 route, triggering the peer VTEP to move from non-DF to DF. The new DF now starts sending LACP *in-sync* PDUs, causing the connected server interface to be bundled back into the LAG, which can now actively forward traffic.

Like all-active, the configuration of single-active ES-based LAG includes the following:

- mapping the physical interface to a LAG interface
- configuring the LAG interface with required LACP parameters
- configuring subinterfaces within the LAG interface to accept tagged or untagged Layer 2 packets as required
- configuring an Ethernet segment (and all required parameters) and mapping it to the respective LAG interface (notably, the *multi-homing-mode* configuration option is set to *single-active*)
- configuring the Ethernet segment on the active node with a higher preference and using a preference-based algorithm for DF election
- configuring *interface-standby-signaling-on-non-df* (under the *df-election hierarchy*) to send a LACP out-of-sync on non-DF nodes, keeping the server links connected to the non-DF nodes in a *down* state
- setting the DF election activation timer to 0 (the default timer is 3 seconds); this timer controls the delay of transition from non-DF to DF

```
// physical interface mapped to LAG interface

A:admin@d3l-29-spine1# info interface ethernet-1/23
    description spine1-spine2-tor4-lag
    admin-state enable
    ethernet {
        aggregate-id lag4
        lacp-port-priority 32768
        reload-delay 100
    }

// configuration of LAG interface

A:admin@d3l-29-spine1# info interface lag4
    description spine1-spine2-tor4-lag
    admin-state enable
    vlan-tagging true
    ethernet {
        standby-signaling lacp
    }
    subinterface 40 {
        type bridged
        description tagged-v40
        admin-state enable
        vlan {
            encap {
                single-tagged {
                    vlan-id 40
                }
            }
        }
    }
    subinterface 4096 {
        type bridged
        description untagged-v10
        admin-state enable
        vlan {
            encap {
                untagged {
                }
            }
        }
    }
    lag {
        lag-type lacp
        min-links 1
        lacp-fallback-mode static
        lacp-fallback-timeout 60
        lacp {
            interval FAST
            lacp-mode ACTIVE
            admin-key 7
            system-id-mac 00:00:12:00:00:00
            system-priority 32768
        }
    }

// Ethernet segment configuration on active VTEP

A:admin@d3l-29-spine1# info system network-instance protocols evpn ethernet-segments bgp-
instance 1 ethernet-segment spine1-spine2-tor4-lag
```

```
    admin-state enable
    esi 00:00:00:12:00:00:00:00:00:00
    multi-homing-mode single-active
    interface lag4 {
    }
    df-election {
        timers {
            activation-timer 0
        }
        interface-standby-signaling-on-non-df {
        }
        algorithm {
            type preference
            preference-alg {
                preference-value 800
                capabilities {
                    non-revertive true
                }
            }
        }
    }

// Ethernet segment configuration on standby VTEP

A:admin@d3l-30-spine2# info system network-instance protocols evpn ethernet-segments bgp-
instance 1 ethernet-segment spine1-spine2-tor4-lag
    admin-state enable
    esi 00:00:00:12:00:00:00:00:00:00
    multi-homing-mode single-active
    interface lag4 {
    }
    df-election {
        timers {
            activation-timer 0
        }
        interface-standby-signaling-on-non-df {
        }
        algorithm {
            type preference
            preference-alg {
                preference-value 500
                capabilities {
                    non-revertive true
                }
            }
        }
    }

// LAG interface state on active VTEP

A:admin@d3l-29-spine1# show lag lag4 lacp-state | as yaml
---
LacpHeader:
  - Lag Id: lag4
    LacpBrief:
      Interval: FAST
      Mode: ACTIVE
      System Id: '00:00:12:00:00:00'
      System Priority: 32768
    LacpState:
      - Members: ethernet-1/23
        Oper state: up
        Activity: ACTIVE
```

```
        Timeout: SHORT
        State: IN_SYNC/True/True/True
        System Id: '00:00:12:00:00:00'
        Oper key: 7
        Partner Id: '00:00:00:00:00:01'
        Partner Key: 1
        Port No: 1
        Partner Port No: 1

// LAG interface state on standby VTEP

A:admin@d31-30-spine2# show lag lag4 lacp-state | as yaml
---
LacpHeader:
  - Lag Id: lag4
    LacpBrief:
      Interval: FAST
      Mode: ACTIVE
      System Id: '00:00:12:00:00:00'
      System Priority: 32768
    LacpState:
      - Members: ethernet-1/23
        Oper state: down(lacp-down)
        Activity: ACTIVE
        Timeout: SHORT
        State: OUT_SYNC/True/False/False
        System Id: '00:00:12:00:00:00'
        Oper key: 7
        Partner Id: '00:00:00:00:00:01'
        Partner Key: 1
        Port No: 1
        Partner Port No: 1
```

Example 9 **Configuration of single-active ES-based LAG**

4.10 LAG interfaces on ToRs

The ToR switches are configured with link aggregation groups using LACP for interfaces connecting to the collapsed spines.

```
// LAG on ToR switches

A:admin@d21-34-tor1# info interface ethernet-1/{51,52}
    interface ethernet-1/51 {
        description tor1-spine-lag
        admin-state enable
        ethernet {
            aggregate-id lag1
            lacp-port-priority 32768
        }
    }
    interface ethernet-1/52 {
        description tor1-spine-lag
        admin-state enable
        ethernet {
            aggregate-id lag1
            lacp-port-priority 32768
        }
```

```
    }

A:admin@d21-34-tor1# info interface lag1
    description tor1-spine-lag
    admin-state enable
    vlan-tagging true
    subinterface 50 {
        type bridged
        description tagged-v50-tor
        admin-state enable
        vlan {
            encap {
                single-tagged {
                    vlan-id 50
                }
            }
        }
    }
    lag {
        lag-type lacp
        min-links 1
        lacp-fallback-mode static
        lacp-fallback-timeout 60
        lacp {
            interval FAST
            lacp-mode ACTIVE
            admin-key 6
            system-id-mac 00:00:11:11:11:11
            system-priority 32768
        }
    }
```

Example 10 **Configuration of LAG interface on ToR switches**

4.11 Layer 3 server-facing interfaces

Layer 3 server-facing interfaces are commonly deployed for cloud-native environments, enabling an end-to-end routing design. While the NVD is tested by simply exporting the directly connected Layer 3 interface subnet for reachability to the server (these Layer 3 servers connect directly to the collapsed spines), you can also choose to run BGP between the collapsed spine and the server for dynamic exchange of routes (these received routes are then exported as EVPN Type-5 routes for connectivity).

In this case, the Layer 3 interface is created as a tagged subinterface with an IPv4/IPv6 address assigned to it. It is also mapped to its respective IP VRF.

```
// Layer 3 subinterface created on spine1

A:admin@d31-29-spine1# info interface ethernet-1/27 subinterface 100
    type routed
    description spine1-l3-1
    admin-state enable
    ip-mtu 9200
    ipv4 {
        admin-state enable
        address 172.16.100.2/31 {
```

```
            primary
        }
        arp {
            timeout 14400
        }
    }
    ipv6 {
        admin-state enable
        address 2001:db8:0:100::2/127 {
            primary
        }
    }
    vlan {
        encap {
            single-tagged {
                vlan-id 100
            }
        }
    }

A:admin@d3l-29-spine1# info network-instance vrf1
    type ip-vrf
    admin-state enable
    description vrf1
    interface ethernet-1/27.100 {
    }

*snip*
```

Example 11 **Configuration of Layer 3 server-facing interface**

4.12 IRB interfaces

```
// IRB subinterface on spine1

A:admin@d3l-29-spine1# info interface irb0 subinterface 0
    description irb-v10
    ip-mtu 9200
    ipv4 {
        admin-state enable
        address 172.16.10.254/24 {
            anycast-gw true
            primary
        }
        arp {
            timeout 250
            learn-unsolicited true
            proxy-arp true
            evpn {
                advertise dynamic {
                }
            }
        }
    }
    ipv6 {
        admin-state enable
        address 2001:db8:0:10::254/64 {
            anycast-gw true
            primary
```

```
        }
        neighbor-discovery {
            learn-unsolicited both
            proxy-nd true
            evpn {
                advertise dynamic {
                }
            }
        }
    }
    anycast-gw {
        virtual-router-id 1
    }
```

Example 12 **Configuration of an IRB subinterface on spine1**

4.13 VXLAN tunnels

VXLAN tunnels are created as tunnel interfaces on SR Linux, where each subinterface is mapped to a bridged VNI (L2VNI) or routed VNI (L3VNI). A sample configuration is provided below, demonstrating a bridged tunnel and a routed tunnel. These bridged and routed tunnel interfaces are associated to their corresponding network instances—bridged VXLAN tunnel interfaces for MAC VRFs (Layer 2) and routed VXLAN interfaces to IP VRFs (Layer 3).

```
// Bridged and routed VXLAN tunnels

A:admin@d3l-29-spine1# info tunnel-interface vxlan0 vxlan-interface {500,501}
    vxlan-interface 500 {
        type routed
        ingress {
            vni 10500
        }
        egress {
            source-ip use-system-ipv4-address
        }
    }
    vxlan-interface 501 {
        type bridged
        ingress {
            vni 10010
        }
        egress {
            source-ip use-system-ipv4-address
        }
    }
```

Example 13 **Configuration of bridged and routed VXLAN tunnel interfaces on collapsed spines**

4.14 MAC VRFs

MAC VRFs are created for Layer 2 isolation. These MAC VRFs are mapped to a bridged VXLAN tunnel-interface and the bridge domains' corresponding IRB interface, along with the required Layer 2 server-facing subinterfaces (these can be subinterfaces of a physical or LAG interface). Every

MAC VRF is associated with a corresponding import and export route target, which facilitates the import and export of BGP EVPN routes for this MAC VRF. In addition, MAC VRFs are configured with the following options:

- For overlay ECMP, the *ecmp* configuration option is used and set to a value of 8.
- The configuration option *advertise-arp-nd-only-with-mac-table-entry* is set to *true*. This is necessary for multihoming segments, without which misleading MAC mobility events might occur, and especially for single-active Ethernet segments, where a transition from non-DF to DF may impact traffic convergence without configuration option set to *true*.
- Each MAC VRF is enabled with the default duplicate MAC detection timers.

```
// VLAN-based MAC VRF configuration on spine1

A:admin@d3l-29-spine1# info network-instance macvrf-v10
    type mac-vrf
    admin-state enable
    description macvrf-v10
    interface ethernet-1/27.4096 {
    }
    interface irb0.0 {
    }
    interface lag2.4096 {
    }
    interface lag4.4096 {
    }
    vxlan-interface vxlan0.501 {
    }
    protocols {
        bgp-evpn {
            bgp-instance 1 {
                vxlan-interface vxlan0.501
                evi 10
                ecmp 8
                routes {
                    bridge-table {
                        mac-ip {
                            advertise-arp-nd-only-with-mac-table-entry true
                        }
                    }
                }
            }
        }
        bgp-vpn {
            bgp-instance 1 {
                route-target {
                    export-rt target:1:10
                    import-rt target:1:10
                }
            }
        }
    }
    bridge-table {
        mac-learning {
            admin-state enable
```

```
            aging {
                admin-state enable
                age-time 300
            }
        }
        mac-duplication {
            admin-state enable
            monitoring-window 3
            num-moves 5
            hold-down-time 9
            action stop-learning
        }
    }
```

Example 14 **Configuration of a MAC VRF on spine1**

4.15 IP VRFs

IP VRFs are used for Layer 3 isolation and to enable the use of a common, physical infrastructure for multiple, logically isolated tenants or services. The respective IRB subinterfaces are mapped to their corresponding IP VRFs along with a routed VXLAN tunnel-interface (which is the L3VNI for that IP VRF).

As for MAC VRFs, each IP VRF is associated with an export and import route target.

```
// IP VRF configuration on spine1

A:admin@d31-29-spine1# info network-instance vrf1
    type ip-vrf
    admin-state enable
    description vrf1
    interface ethernet-1/27.100 {
    }
    interface irb0.0 {
    }
    interface irb0.1 {
    }
    interface irb0.2 {
    }
    interface irb0.3 {
    }
    interface irb0.5 {
    }
    vxlan-interface vxlan0.500 {
    }
    protocols {
        bgp-evpn {
            bgp-instance 1 {
                vxlan-interface vxlan0.500
                evi 500
                ecmp 8
                routes {
                    route-table {
                        mac-ip {
                            advertise-gateway-mac true
                        }
                    }
                }
```

```
            }
        }
        bgp-vpn {
            bgp-instance 1 {
                route-target {
                    export-rt target:1:500
                    import-rt target:1:500
                }
            }
        }
    }
```

Example 15 **Configuration of an IP VRF on spine1**

5 Test summary

5.1 Feature validation matrix

Feature	SRL v25.3.2	EDA v25.4.1	
		Validation	Configlet required
IPv6 link-local addressing with IPv6 ND for fabric underlay	Validated	Validated	No
Advertise and receive BGP IPv4 NLRIs with IPv6 next hops (RFC 8950)	Validated	Validated	No
Logging in SR Linux for different subsystems	Validated	Validated	Yes
MP-BGP style eBGP peering for underlay and overlay routes	Validated	Validated	No
2-byte BGP ASN support	Validated	Validated	No
Routing policies for underlay and overlay	Validated	Validated	No
Sub-second BFD convergence (750 ms)	Validated	Validated	No
LLDP	Validated	Validated	No
Layer 2 untagged server-facing interfaces	Validated	Validated	No
Layer 2 tagged server-facing interfaces	Validated	Validated	No
Layer 3 server-facing interfaces with IPv4 and IPv6 addressing	Validated	Validated	No
System-wide MTU	Validated	Validated	Yes
Anycast GWs with IPv4 and IPv6 addressing	Validated	Validated	No
ESI-based LAG in all-active mode	Validated	Validated	No
ESI-based LAG in single-active mode	Validated	Validated	No
ECMP for underlay and overlay	Validated	Validated	No
Asymmetric IRB routing	Validated	Validated	No
Symmetric IRB routing with Type-5	Validated	Validated	No
VLAN-based MAC VRFs	Validated	Validated	Yes

Feature	SRL v25.3.2	EDA v25.4.1	
		Validation	Configlet required
IP VRFs	Validated	Validated	No
gNMI-based telemetry	Validated	Validated	No

Table 2 **Feature matrix**

5.2 Traffic convergence results

Test	Description	Approximate convergence time
Link shut	Traffic flows are enabled between single-homed servers connected directly to the collapsed spines, link is shut down, and traffic convergence is measured for IPv4 and IPv6 traffic	0.0134s
BGP reset	Traffic flows are enabled between single-homed servers connected directly to the collapsed spines, BGP peering is reset using CLI, and traffic convergence is measured for IPv4 and IPv6 traffic	0.001s
Ethernet segments in all-active mode	ES-based LAG is configured in all-active mode and traffic flows are enabled that traverse this segment using local bias forwarding. During this state, the local exit interface is shut, and convergence time is measured (for traffic to move to remote VTEP) for IPv4 and IPv6 traffic	0.0129s
Ethernet segments in single-active with LACP signaling off on non-DF	ES-based LAG is configured in single-active mode with LACP signaling off on non-DF (port active mode) with traffic flowing over the local interface. During this steady state, the local exit interface is shut and traffic convergence is measured (non-DF will transition to DF for traffic forwarding)	0.215s
ToR uplink failure	ToR uplink (part of LAG) is shut and IPv4 and IPv6 traffic convergence is measured	0.005s
ToR switch reboot	ToR switch is rebooted while traffic is flowing through the switch	99s
Spine reboot	Remote spine is rebooted while traffic is flowing through it for directly attached servers	108s

Table 3 **Traffic convergence metrics**

6 EDA integration

6.1 EDA architecture

Nokia's Event Driven Automation (EDA) platform is a cloud-native platform deployed on top of Kubernetes, leveraging the Kubernetes-provided declarative API, tooling, and the ecosystem around it. EDA can be deployed as a single or multimode cluster.

The various components of the EDA/K8s tech stack are shown below, instantiated as Kubernetes pods.

```
admin@server:~/nvd-hw/collapsed-spine$ kubectl get pods -A
NAMESPACE        NAME                                        READY   STATUS      RESTARTS
AGE
cert-manager     cert-manager-777c6f8ff4-bscqv               1/1     Running     0
6d11h
cert-manager     cert-manager-cainjector-6558fc6578-6clk2    1/1     Running     0
6d11h
cert-manager     cert-manager-webhook-6964489477-khzbx       1/1     Running     0
6d11h
eda-system       cert-manager-csi-driver-4b7mx               3/3     Running     0
6d11h
eda-system       cert-manager-csi-driver-8279p               3/3     Running     0
6d11h
eda-system       cert-manager-csi-driver-ct2wk               3/3     Running     0
6d11h
eda-system       cert-manager-csi-driver-cttlg               3/3     Running     0
6d11h
eda-system       cert-manager-csi-driver-mznkr               3/3     Running     0
6d11h
eda-system       cert-manager-csi-driver-tsvnq               3/3     Running     0
6d11h
eda-system       eda-api-bdd576c85-l2252                     1/1     Running     0
6d11h
eda-system       eda-appstore-74cdc5c964-csf8j               1/1     Running     0
6d11h
eda-system       eda-asvr-9fd4b99fb-lxvwh                    1/1     Running     0
6d11h
eda-system       eda-bsvr-b7b84b8f5-5689z                    1/1     Running     0
6d11h
eda-system       eda-ce-58c5cbf87d-cwmmq                     1/1     Running     0
5d23h
eda-system       eda-cert-checker-bf74ccbd4-m48bn            1/1     Running     0
6d11h
eda-system       eda-fe-b8b877cf6-ntc47                      1/1     Running     0
6d11h
eda-system       eda-fluentbit-4mz4x                         1/1     Running     0
6d11h
eda-system       eda-fluentbit-5mszs                         1/1     Running     0
6d11h
eda-system       eda-fluentbit-6ksw7                         1/1     Running     0
6d11h
eda-system       eda-fluentbit-95z2p                         1/1     Running     0
6d11h
eda-system       eda-fluentbit-j6swr                         1/1     Running     0
6d11h
eda-system       eda-fluentbit-q4smx                         1/1     Running     0
6d11h
eda-system       eda-fluentd-7cd48db9c5-rxs9d                1/1     Running     0
6d11h
```

```
eda-system       eda-git-5db9dfc7bc-pdb6m                          1/1     Running     0
6d11h
eda-system       eda-git-replica-f69b9c9f4-jznkq                   1/1     Running     0
6d11h
eda-system       eda-keycloak-bcfbfd9d6-bn7mn                      1/1     Running     0
5d23h
eda-system       eda-metrics-server-8d8b8595f-r6hnd                1/1     Running     0
6d11h
eda-system       eda-npp-0                                         1/1     Running     0
5d22h
eda-system       eda-npp-1                                         1/1     Running     0
5d20h
eda-system       eda-postgres-6b59c9985-ppbkf                      1/1     Running     0
6d11h
eda-system       eda-px-55c6dd5588-cfvw8                           1/1     Running     0
43h
eda-system       eda-sa-54df7ffbc5-xjbht                           1/1     Running     0
6d11h
eda-system       eda-sc-7644c9cc4c-kj7xm                           1/1     Running     0
6d11h
eda-system       eda-se-1                                          1/1     Running     0
6d11h
eda-system       eda-toolbox-696f47749d-z457h                      1/1     Running     0
6d11h
eda-system       trust-manager-849b644bdf-88pck                    1/1     Running     0
6d11h
kube-system      coredns-578d4f8ffc-jdqq7                          1/1     Running     0
6d11h
kube-system      coredns-578d4f8ffc-n4c5n                          1/1     Running     0
6d11h
kube-system      kube-apiserver-eda-11                             1/1     Running     0
6d11h
kube-system      kube-apiserver-eda-12                             1/1     Running     0
6d11h
kube-system      kube-apiserver-eda-13                             1/1     Running     0
6d11h
kube-system      kube-controller-manager-eda-11                    1/1     Running     1 (6d11h ago)
6d11h
kube-system      kube-controller-manager-eda-12                    1/1     Running     0
6d11h
kube-system      kube-controller-manager-eda-13                    1/1     Running     0
6d11h
kube-system      kube-flannel-6b9vq                                1/1     Running     0
6d11h
kube-system      kube-flannel-7xd69                                1/1     Running     0
6d11h
kube-system      kube-flannel-qcd26                                1/1     Running     0
6d11h
kube-system      kube-flannel-qzkm4                                1/1     Running     0
6d11h
kube-system      kube-flannel-ss7qn                                1/1     Running     0
6d11h
kube-system      kube-flannel-xcwct                                1/1     Running     0
6d11h
kube-system      kube-proxy-4mf92                                  1/1     Running     0
6d11h
kube-system      kube-proxy-jjkm5                                  1/1     Running     0
6d11h
kube-system      kube-proxy-p9mfg                                  1/1     Running     0
6d11h
kube-system      kube-proxy-phtxl                                  1/1     Running     0
6d11h
kube-system      kube-proxy-q566g                                  1/1     Running     0
6d11h
kube-system      kube-proxy-vpxzc                                  1/1     Running     0
6d11h
kube-system      kube-scheduler-eda-11                             1/1     Running     1 (6d11h ago)
6d11h
kube-system      kube-scheduler-eda-12                             1/1     Running     0
6d11h
```

```
kube-system      kube-scheduler-eda-13                                1/1     Running     0
6d11h
metallb-system   controller-5cbffbc46b-xwzz8                          1/1     Running     0
6d11h
metallb-system   speaker-6lpjs                                        1/1     Running     0
6d11h
metallb-system   speaker-cbclx                                        1/1     Running     0
6d11h
metallb-system   speaker-mwvfw                                        1/1     Running     0
6d11h
metallb-system   speaker-pwhqn                                        1/1     Running     0
6d11h
metallb-system   speaker-pwrrt                                        1/1     Running     0
6d11h
metallb-system   speaker-rc9p2                                        1/1     Running     0
6d11h
rook-ceph        csi-cephfsplugin-d8vdb                               2/2     Running     1 (6d11h ago)
6d11h
rook-ceph        csi-cephfsplugin-f25fm                               2/2     Running     1 (6d11h ago)
6d11h
rook-ceph        csi-cephfsplugin-p5pxz                               2/2     Running     1 (6d11h ago)
6d11h
rook-ceph        csi-cephfsplugin-p92lt                               2/2     Running     1 (6d11h ago)
6d11h
rook-ceph        csi-cephfsplugin-provisioner-5b8485cdb4-4v8wp        5/5     Running     2 (6d11h ago)
6d11h
rook-ceph        csi-cephfsplugin-provisioner-5b8485cdb4-zzrtr        5/5     Running     1 (6d11h ago)
6d11h
rook-ceph        csi-cephfsplugin-t4bpg                               2/2     Running     1 (6d11h ago)
6d11h
rook-ceph        csi-cephfsplugin-w7mzc                               2/2     Running     1 (6d11h ago)
6d11h
rook-ceph        rook-ceph-mds-ceph-filesystem-a-649b4d448-525fp      1/1     Running     0
6d11h
rook-ceph        rook-ceph-mds-ceph-filesystem-b-6fd79c7d47-g42d2     1/1     Running     0
6d11h
rook-ceph        rook-ceph-mgr-a-6cb6b8d4fb-v2r7r                     2/2     Running     0
6d11h
rook-ceph        rook-ceph-mgr-b-7896d689d8-9wspm                     2/2     Running     0
6d11h
rook-ceph        rook-ceph-mon-a-74688f9f97-n7vqz                     1/1     Running     0
6d11h
rook-ceph        rook-ceph-mon-b-bd754b95f-2cs5z                      1/1     Running     0
6d11h
rook-ceph        rook-ceph-mon-c-6d9b6cbdf6-ggjjp                     1/1     Running     0
6d11h
rook-ceph        rook-ceph-operator-6c99bbf54d-g7xpw                  1/1     Running     0
6d11h
rook-ceph        rook-ceph-osd-0-bb8968b97-r49z6                      1/1     Running     0
6d11h
rook-ceph        rook-ceph-osd-1-554cb45854-9z8g6                     1/1     Running     0
6d11h
rook-ceph        rook-ceph-osd-2-5cf9747c5-95rvq                      1/1     Running     0
6d11h
rook-ceph        rook-ceph-osd-prepare-eda-11-kbzsz                   0/1     Completed   0
6d11h
rook-ceph        rook-ceph-osd-prepare-eda-12-j6ftd                   0/1     Completed   0
6d11h
rook-ceph        rook-ceph-osd-prepare-eda-13-jbrsb                   0/1     Completed   0
6d11h
rook-ceph        rook-ceph-osd-prepare-eda-14-77km8                   0/1     Completed   0
6d11h
rook-ceph        rook-ceph-osd-prepare-eda-15-56d7r                   0/1     Completed   0
6d11h
rook-ceph        rook-ceph-osd-prepare-eda-16-r6sxr                   0/1     Completed   0
6d11h
rook-ceph        rook-ceph-tools-54bcc747b4-xk8rx                     1/1     Running     0
6d11h
```

Example 16 **EDA namespaces and pods**

Some commonly used pods and their functionalities are:

- **eda-asvr** - the artifact server stores common artifacts used in EDA functionality. Examples include SRLinux image, SRL MD5 hash, yang path.zip, and so forth. The availability of an artifact can be verified with “kubectl get artifacts -A”.
- **eda-bsvr** – the bootstrap server is responsible for all onboarding of nodes (virtual or hardware). Onboarding involves gNMI discovery, gNMI management, and instantiation of NPP pods for node lifecycle management.
- **eda-ce** – the configuration engine keeps track of all the dependencies among the application resources and runs the application intents when needed.
- **eda-npp** – the eda-npp pod is responsible for schema validation of the generated configuration. Additionally, it is responsible for all communications to the devices for both setting configuration and retrieving state.
- **eda-api** – the eda-api pod is the REST API server, which is accessible to end users and is consumed by the GUI.
- **eda-cx** – the sandbox controller spins up simulated nodes for building digital twins of the fabric (the example above has the mode set to physical hardware only, hence the EDA CX functionality has been disabled)
- **eda-toolbox** – the toolbox provides tools such as *edactl* for insight into EDA transactions and an EDA topology generator that can generate a topology from a YAML file

Figure 15 demonstrates the high-level workflow required to build the prescriptive collapsed spine NVD. The resources shown can be created using either the EDA UI or natively using Kubernetes manifest files.

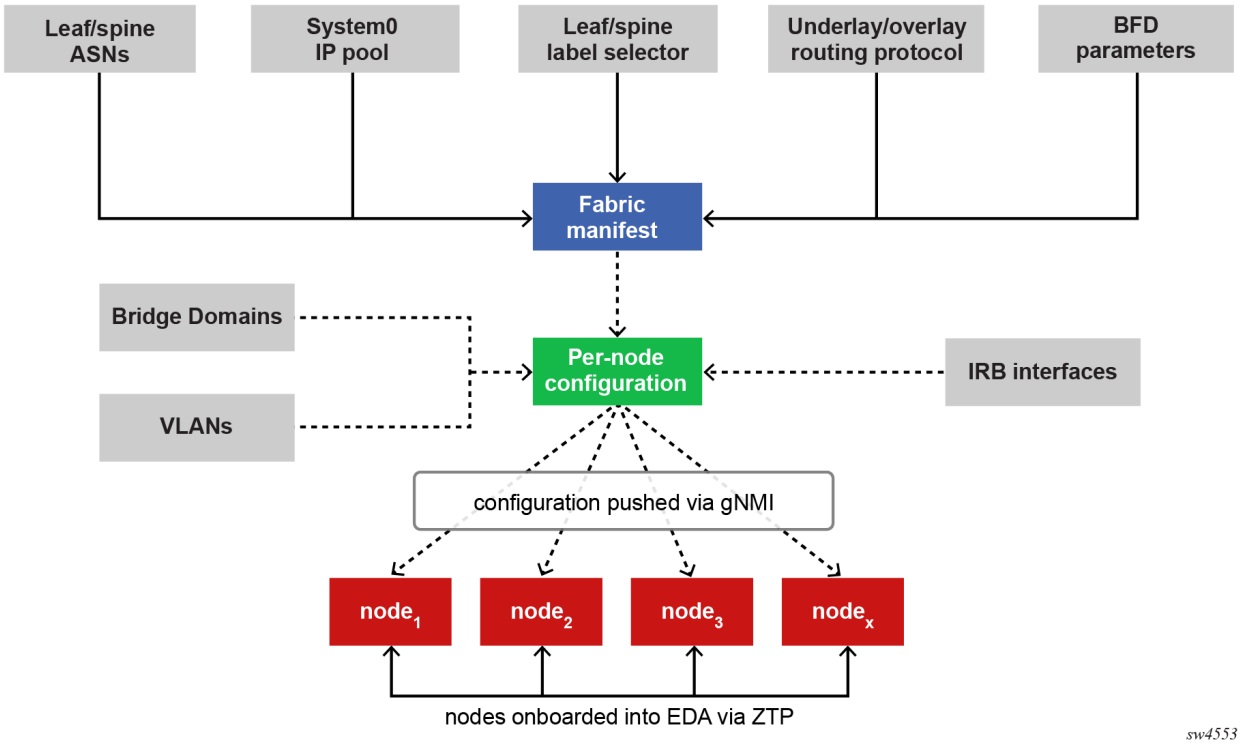

Figure 15 **High-level EDA workflow to deploy a fabric**

After this workflow is completed with all nodes onboarded and the fabric fully deployed, the topology can be viewed by navigating to **Main** → **Topologies** → **Physical**. See Figure 16 for reference.

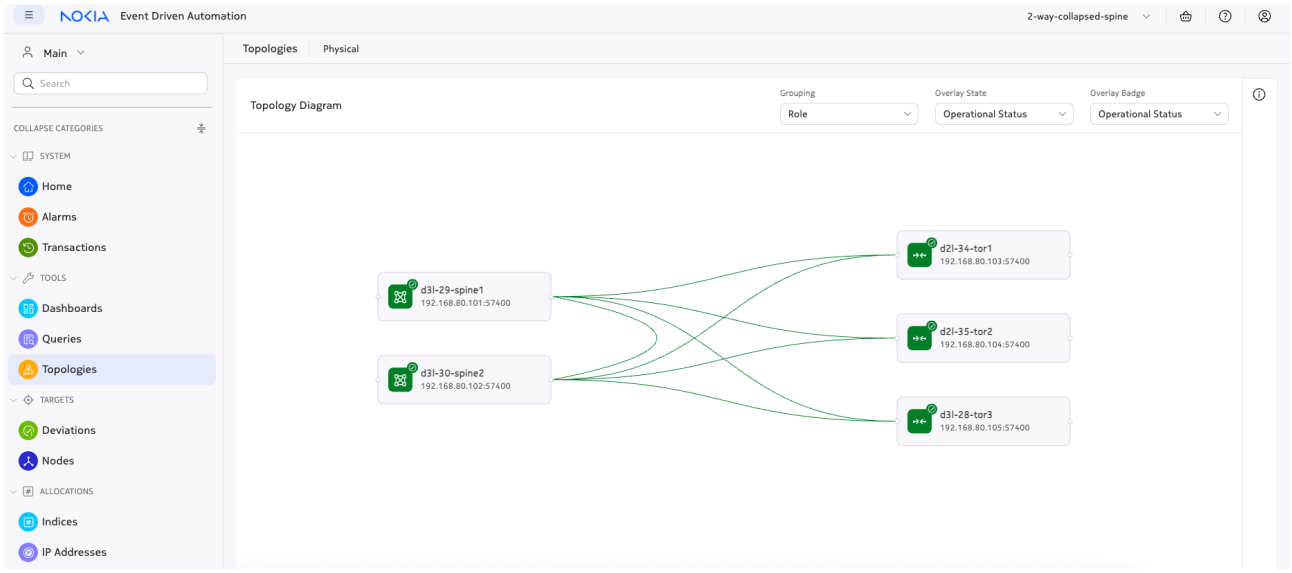

Figure 16 **Collapsed spine EVPN VXLAN NVD fabric onboarded and deployed in EDA**

6.2 EDA onboarding with ZTP

EDA has the capability to onboard fabric nodes via ZTP. EDA, as the ZTP server, can fully automate the end-to-end deployment of Nokia SR Linux nodes. Nodes which are in a factory default state only need to be plugged into the out-of-band (OOB) infrastructure and EDA can onboard the devices, along with pushing expected configuration (based on user intent) to them.

Figure 17 provides a high-level overview of the ZTP workflow and Example 17 displays console logs from a Nokia 7220 IXR-D3L being onboarded.

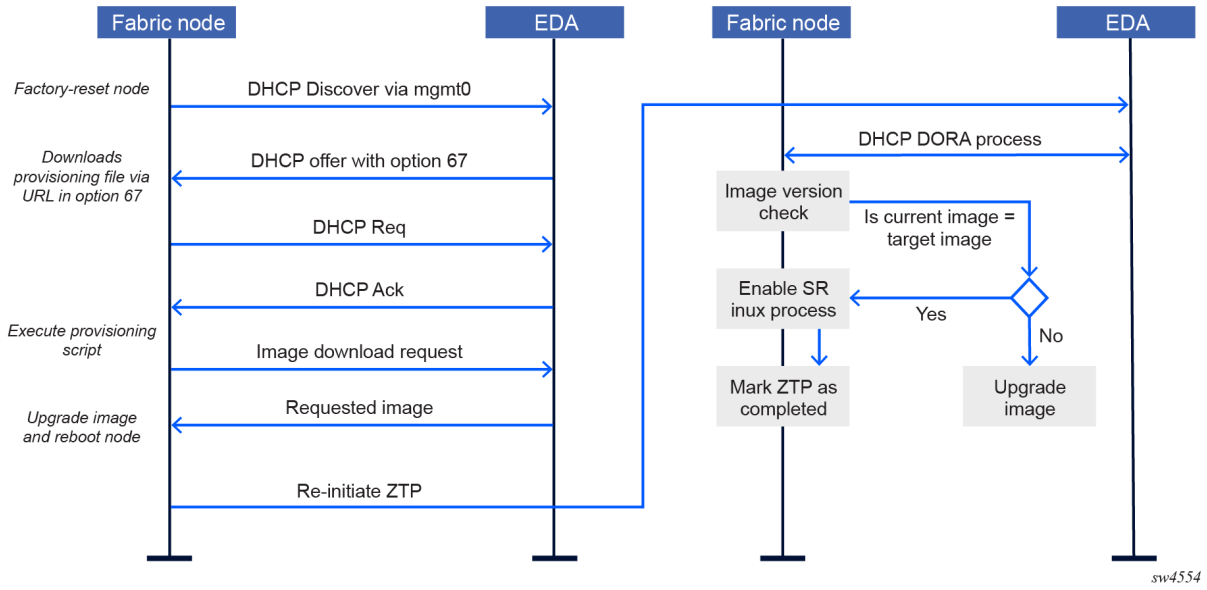

Figure 17 **ZTP workflow**

```
[linuxadmin@srlinux ~]$ ztp service restart --autoboot enable
Warning: This command to be used under guidance of Nokia Technical Support only.
Unsupported usage can trigger instability of network
+--------+-----------------+
| key    | value           |
+--------+-----------------+
| status | Service started |
+--------+-----------------+
2025:06:26 10:32:30:78 | EVENT | Starting ZTP process
2025:06:26 10:32:30:82 | EVENT | Current version 25.3.2-314
2025:06:26 10:32:36:00 | EVENT | ZTP runtime remaining 3594.73 seconds
2025:06:26 10:32:36:02 | EVENT | ZTP Perform DHCP_V4
2025:06:26 10:32:41:38 | EVENT | Received dhcp lease on mgmt0 for 192.168.80.102/24, from
server 192.168.80.7
2025:06:26 10:32:41:47 | EVENT | option 67 provided by dhcp:
http://192.168.80.7:80/core/httpproxy/v1/asvr/2-way-collapsed-spine/init-base/bootscript-
d3l-30-spine2/d3l-30-spine2-provision.py
2025:06:26 10:32:41:66 | EVENT | Updated hostname to d3l-30-spine2
2025:06:26 10:32:41:67 | EVENT | option 67 provided by dhcp:
http://192.168.80.7:80/core/httpproxy/v1/asvr/2-way-collapsed-spine/init-base/bootscript-
d3l-30-spine2/d3l-30-spine2-provision.py
2025:06:26 10:32:41:68 | EVENT | Url to fetch provisioning script:
http://192.168.80.7:80/core/httpproxy/v1/asvr/2-way-collapsed-spine/init-base/bootscript-
d3l-30-spine2/d3l-30-spine2-provision.py
```

```
2025:06:26 10:32:41:68 | EVENT | Executing provisioning script
2025:06:26 10:32:41:76 | EVENT | Downloaded provisioning script to
/etc/opt/srlinux/ztp/script/provision.py
2025:06:26 10:34:23:15 | EVENT | Installing image. Url:
https://192.168.80.7:443/core/httpproxy/v1/asvr/2-way-collapsed-spine/srlimages/srlinux-
25.3.2-bin/srlinux.bin
2025:06:26 10:34:34:33 | EVENT | Version of new image 25.3.2-314
2025:06:26 10:34:34:33 | EVENT | Current version: 25.3.2-314, New version: 25.3.2-314
2025:06:26 10:34:34:34 | EVENT | New image version 25.3.2-314 is same as active version
25.3.2-314
2025:06:26 10:34:34:34 | EVENT | Not performing image upgrade
2025:06:26 10:34:38:64 | EVENT | Reboot will be triggered
2025:06:26 10:34:38:70 | EVENT | Chassis reboot requested.
2025:06:26 10:34:40:98 | EVENT | sr_cli chassis reboot force requested.
25-06-26 10:34:42.596 sr_device_mgr: Chassis reboot force requested - rebooting
25-06-26 10:34:43.646 sr_device_mgr: Rebooting - chassis reboot requested

*snip*

// after it reboots, same process again with ZTP completion

2025:06:26 10:36:04:08 | EVENT | ZTP runtime remaining 3591.47 seconds
2025:06:26 10:36:04:10 | EVENT | ZTP Perform DHCP_V4

2025:06:26 10:36:04:43 | EVENT | Received dhcp lease on mgmt0 for 192.168.80.102/24, from
server 192.168.80.7
2025:06:26 10:36:04:50 | EVENT | option 67 provided by dhcp:
http://192.168.80.7:80/core/httpproxy/v1/asvr/2-way-collapsed-spine/init-base/bootscript-
d3l-30-spine2/d3l-30-spine2-provision.py
2025:06:26 10:36:04:71 | EVENT | Updated hostname to d3l-30-spine2
2025:06:26 10:36:04:71 | EVENT | option 67 provided by dhcp:
http://192.168.80.7:80/core/httpproxy/v1/asvr/2-way-collapsed-spine/init-base/bootscript-
d3l-30-spine2/d3l-30-spine2-provision.py
2025:06:26 10:36:04:72 | EVENT | Url to fetch provisioning script:
http://192.168.80.7:80/core/httpproxy/v1/asvr/2-way-collapsed-spine/init-base/bootscript-
d3l-30-spine2/d3l-30-spine2-provision.py
2025:06:26 10:36:04:72 | EVENT | Executing provisioning script
2025:06:26 10:36:04:79 | EVENT | Downloaded provisioning script to
/etc/opt/srlinux/ztp/script/provision.py
2025:06:26 10:36:06:46 | EVENT | Starting srlinux
2025:06:26 10:36:13:75 | EVENT | Srlinux process state: running
2025:06:26 10:36:14:78 | EVENT | Srlinux is running
2025:06:26 10:36:22:64 | EVENT | Execution of /etc/opt/srlinux/ztp/script/provision.py
completed with exit code 0
2025:06:26 10:36:22:65 | EVENT | Provisioning script execution successful
2025:06:26 10:36:35:05 | EVENT | Completed ZTP process
```

Example 17 **Console logs on a Nokia 7220 IXR-D3L during successful ZTP onboarding**

6.3 EDA Kubernetes workflow for NVD deployment

This section describes various manifest files that can be used to deploy an EDA-orchestrated collapsed spine EVPN VXLAN fabric in accordance with the prescriptive validated design described in this document. It is important to note that these manifest files must reference the correct EDA namespace (for example, in this case, the namespace *2-way-collapsed-spine* is used).

6.3.1 EDA artifacts for SR Linux version 25.3.2

Kubernetes artifacts are created for target SR Linux version and used in the EDA node profile, Custom Resource. This includes the creation of manifest files for the .bin image, the md5 hash file, and the YAML zip file, samples of which are shown below.

```
# artifacts for SRL v25.3.2
apiVersion: artifacts.eda.nokia.com/v1
kind: Artifact
metadata:
  name: srlinux-25.3.2-bin
  namespace: 2-way-collapsed-spine
spec:
  repo: srlimages
  filePath: srlinux.bin
  remoteFileUrl:
    fileUrl: https://repo.ncse.io/images/srl/srlinux-25.3.2-314.bin
---
apiVersion: artifacts.eda.nokia.com/v1
kind: Artifact
metadata:
  name: srlinux-25.3.2-md5
  namespace: 2-way-collapsed-spine
spec:
  repo: srlimages
  filePath: srlinux.md5
  remoteFileUrl:
    fileUrl: https://repo.ncse.io/images/srl/srlinux-25.3.2-314.bin.md5
---
# yang path must point to EDA-specific YANG zip file
apiVersion: artifacts.eda.nokia.com/v1
kind: Artifact
metadata:
  name: srlinux-25.3.2
  namespace: 2-way-collapsed-spine
spec:
  repo: schemaprofiles
  filePath: srlinux-25.3.2.zip
  remoteFileUrl:
    fileUrl: https://repo.ncse.io/images/srl/srlinux-25.3.2-312.zip
```

Example 18 **EDA artifact manifest**

6.3.2 Subnet allocation for management of SR Linux fabric nodes

A manifest file is created to instantiate an IPv4/IPv6 subnet pool for the management of SR Linux fabric nodes. This pool is used to give an IP address to the management interface of the SR Linux

fabric nodes during ZTP by referencing this pool in the node profile manifest (shown in Section 6.3.5).

```
# management pool for ZTP
apiVersion: core.eda.nokia.com/v1
kind: IPInSubnetAllocationPool
metadata:
  name: hw-ipv4-mgmt-pool
  namespace: 2-way-collapsed-spine
spec:
  segments:
    - subnet: 192.168.80.0/24
      allocations:
        - name: gateway$$
          value: 192.168.80.254/24
      reservations:
        - start: 192.168.80.1/24
          end: 192.168.80.100/24
```

Example 19 **EDA subnet and IP pool allocation manifest**

6.3.3 EDA node group creation

An EDA node group determines the level of services a node user has access to. This node group is referenced in a node user (shown in Section 6.3.4).

```
# node group for NVD
apiVersion: v1
items:
- apiVersion: aaa.eda.nokia.com/v1alpha1
  kind: NodeGroup
  metadata:
    name: sudo
    namespace: 2-way-collapsed-spine
  spec:
    services:
    - GNMI
    - CLI
    - NETCONF
    - GNOI
    - GNSI
    superuser: true
kind: List
```

Example 20 **EDA node group manifest**

6.3.4 EDA node user creation

A node user uses the node group to manage access to TopoNodes (which are essentially your fabric nodes onboarded in EDA).

```
# node user for NVD
apiVersion: core.eda.nokia.com/v1
kind: NodeUser
metadata:
  name: admin
  namespace: 2-way-collapsed-spine
spec:
  groupBindings:
    - groups:
        - sudo
      nodeSelector:
        - ""
  password: NokiaSrl1!
  username: admin
```

Example 21 **EDA node user manifest**

6.3.5 EDA node profile for node onboarding

An EDA node profile facilitates the onboarding of fabric nodes, including the username/password for authentication into the node, a DHCP scope for assignment, and image version check (the node profile image is the expected target image).

```
# nodeprofile for v25.3.2
apiVersion: core.eda.nokia.com/v1
kind: NodeProfile
metadata:
  name: real-srlinux-25.3.2
  namespace: 2-way-collapsed-spine
spec:
  dhcp:
    managementPoolv4: hw-ipv4-mgmt-pool
  images:
    - image: 2-way-collapsed-spine/srlimages/srlinux-25.3.2-bin/srlinux.bin
      imageMd5: 2-way-collapsed-spine/srlimages/srlinux-25.3.2-md5/srlinux.md5
  nodeUser: admin
  onboardingUsername: 'admin'
  onboardingPassword: 'NokiaSrl1!'
  operatingSystem: srl
  port: 57400
```

```
  version: 25.3.2
  versionMatch: v25\.3\.2.*
  versionPath: .system.information.version
  yang: https://eda-asvr.eda-system/2-way-collapsed-spine/schemaprofiles/srlinux-25.3.2/srlinux-25.3.2.zip
```

Example 22 **EDA node profile manifest**

6.3.6 Modify existing init-base custom resource

The existing *init-base* custom resource is modified to set commitSave to *true* so that SR Linux fabric nodes save to startup configuration whenever EDA commits any configuration via gNMI.

```
# init modified to save on commit
apiVersion: v1
items:
- apiVersion: bootstrap.eda.nokia.com/v1alpha1
  kind: Init
  metadata:
    name: init-base
    namespace: 2-way-collapsed-spine
  spec:
    commitSave: true
    nodeSelector:
    - ""
kind: List
```

Example 23 **Resource to enable commit save which save configuration to startup on any commit**

6.3.7 Onboarding nodes in EDA with using a TopoNode Custom Resource

SR Linux nodes can be onboarded into EDA using the TopoNode custom resource. This includes the creation of labels as metadata that will be attached to the node (these labels are used as selectors when deploying the fabric), a node profile name, the platform, and serial number of the node.

```
# toponodes for collapsed spine fabric
apiVersion: core.eda.nokia.com/v1
kind: TopoNode
metadata:
  name: d3l-29-spine1
  namespace: 2-way-collapsed-spine
  labels:
```

```
    eda.nokia.com/role: collapsed-spine
    eda.nokia.com/name: d3l-29-spine1
    eda.nokia.com/security-profile: managed
spec:
  platform: 7220 IXR-D3L
  version: 25.3.2
  operatingSystem: srl
  nodeProfile: real-srlinux-25.3.2
  operatingSystem: srl
  platform: "7220 IXR-D3L"
  version: 25.3.2
  serialNumber: <omitted>
  npp:
    mode: normal
---
apiVersion: core.eda.nokia.com/v1
kind: TopoNode
metadata:
  name: d3l-30-spine2
  namespace: 2-way-collapsed-spine
  labels:
    eda.nokia.com/role: collapsed-spine
    eda.nokia.com/name: d3l-30-spine2
    eda.nokia.com/security-profile: managed
spec:
  platform: 7220 IXR-D3L
  version: 25.3.2
  operatingSystem: srl
  nodeProfile: real-srlinux-25.3.2
  operatingSystem: srl
  platform: "7220 IXR-D3L"
  version: 25.3.2
  serialNumber: <omitted>
  npp:
    mode: normal
---
apiVersion: core.eda.nokia.com/v1
kind: TopoNode
metadata:
  name: d2l-34-tor1
  namespace: 2-way-collapsed-spine
  labels:
    eda.nokia.com/role: tor
    eda.nokia.com/name: d2l-34-tor1
    eda.nokia.com/security-profile: managed
spec:
  platform: 7220 IXR-D2L
```

```
  version: 25.3.2
  operatingSystem: srl
  nodeProfile: real-srlinux-25.3.2
  operatingSystem: srl
  platform: "7220 IXR-D2L"
  version: 25.3.2
  serialNumber: <omitted>
  npp:
    mode: normal
---
apiVersion: core.eda.nokia.com/v1
kind: TopoNode
metadata:
  name: d2l-35-tor2
  namespace: 2-way-collapsed-spine
  labels:
    eda.nokia.com/role: tor
    eda.nokia.com/name: d2l-35-tor2
    eda.nokia.com/security-profile: managed
spec:
  platform: 7220 IXR-D2L
  version: 25.3.2
  operatingSystem: srl
  nodeProfile: real-srlinux-25.3.2
  operatingSystem: srl
  platform: "7220 IXR-D2L"
  version: 25.3.2
  serialNumber: <omitted>
  npp:
    mode: normal
---
apiVersion: core.eda.nokia.com/v1
kind: TopoNode
metadata:
  name: d3l-28-tor3
  namespace: 2-way-collapsed-spine
  labels:
    eda.nokia.com/role: tor
    eda.nokia.com/name: d3l-28-tor3
    eda.nokia.com/security-profile: managed
spec:
  platform: 7220 IXR-D3L
  version: 25.3.2
  operatingSystem: srl
  nodeProfile: real-srlinux-25.3.2
  operatingSystem: srl
  platform: "7220 IXR-D3L"
```

```
  version: 25.3.2
  serialNumber: <omitted>
  npp:
    mode: normal
```

Example 24 **TopoNode resource for node metadata**

6.3.8 Building an ASN pool for collapsed spines

The following manifest file demonstrates how ASN pools can be built to be used during fabric deployment.

```
# ASN pool for collapsed spines
apiVersion: core.eda.nokia.com/v1
kind: IndexAllocationPool
metadata:
  name: collapsed-spine-asn
  namespace: 2-way-collapsed-spine
spec:
  segments:
    - start: 65501
      size: 20
```

Example 25 **ASN pool allocation**

6.3.9 System0 IP pool allocation

In an EVPN VXLAN fabric deployed with VTEP functionality on SR Linux nodes, the system0 IP address is used as the VTEP source address by default. The following manifest file demonstrates how an IP allocation pool is created for assignment as system0 IP address on collapsed spines.

```
# pool for system0 address allocation
apiVersion: core.eda.nokia.com/v1
kind: IPAllocationPool
metadata:
  name: system0
  namespace: 2-way-collapsed-spine
spec:
  segments:
    - subnet: 192.0.2.0/24
```

Example 26 **IP pool for system 0 addresses**

6.3.10 Interface creation

The following snippet of a manifest file demonstrates how interfaces are instantiated in EDA per onboarded SR Linux node.

```
# spine to spine interfaces
apiVersion: interfaces.eda.nokia.com/v1alpha1
kind: Interface
metadata:
  labels:
    eda.nokia.com/role: interSwitch
  name: spine1-ethernet-1-31
  namespace: 2-way-collapsed-spine
spec:
  enabled: true
  lldp: true
  members:
    - enabled: true
      interface: ethernet-1-31
      node: d3l-29-spine1
  type: interface
---
apiVersion: interfaces.eda.nokia.com/v1alpha1
kind: Interface
metadata:
  labels:
    eda.nokia.com/role: interSwitch
  name: spine1-ethernet-1-32
  namespace: 2-way-collapsed-spine
spec:
  enabled: true
  lldp: true
  members:
    - enabled: true
      interface: ethernet-1-32
      node: d3l-29-spine1
  type: interface
---
apiVersion: interfaces.eda.nokia.com/v1alpha1
kind: Interface
metadata:
  labels:
    eda.nokia.com/role: interSwitch
  name: spine2-ethernet-1-31
  namespace: 2-way-collapsed-spine
spec:
```

```
  enabled: true
  lldp: true
  members:
    - enabled: true
      interface: ethernet-1-31
      node: d3l-30-spine2
  type: interface
---
apiVersion: interfaces.eda.nokia.com/v1alpha1
kind: Interface
metadata:
  labels:
    eda.nokia.com/role: interSwitch
  name: spine2-ethernet-1-32
  namespace: 2-way-collapsed-spine
spec:
  enabled: true
  lldp: true
  members:
    - enabled: true
      interface: ethernet-1-32
      node: d3l-30-spine2
  type: interface
---

*snip*
```

Example 27 **Interface custom resource in EDA**

6.3.11 Link creation

The following manifest file demonstrates how links are created between two onboarded nodes in EDA.

```
# spine to spine links
apiVersion: core.eda.nokia.com/v1
kind: TopoLink
metadata:
  labels:
    eda.nokia.com/role: interSwitch
  name: spine1-spine2-link1
  namespace: 2-way-collapsed-spine
spec:
  links:
    - local:
        node: d3l-29-spine1
```

```
      interface: ethernet-1-31
      interfaceResource: "spine1-ethernet-1-31"
    remote:
      node: d3l-30-spine2
      interface: ethernet-1-31
      interfaceResource: "spine2-ethernet-1-31"
    type: interSwitch
---
apiVersion: core.eda.nokia.com/v1
kind: TopoLink
metadata:
  labels:
    eda.nokia.com/role: interSwitch
  name: spine1-spine2-link2
  namespace: 2-way-collapsed-spine
spec:
  links:
    - local:
        node: d3l-29-spine1
        interface: ethernet-1-32
        interfaceResource: "spine1-ethernet-1-32"
      remote:
        node: d3l-30-spine2
        interface: ethernet-1-32
        interfaceResource: "spine2-ethernet-1-32"
      type: interSwitch
---
# spine to tor1 links
apiVersion: core.eda.nokia.com/v1
kind: TopoLink
metadata:
  labels:
    eda.nokia.com/role: edge
  name: spine1-tor1
  namespace: 2-way-collapsed-spine
spec:
  links:
    - local:
        node: d3l-29-spine1
        interface: ethernet-1-25
        interfaceResource: "spine1-ethernet-1-25"
      remote:
        node: d2l-34-tor1
        interface: ethernet-1-51
        interfaceResource: "tor1-ethernet-1-51"
      type: interSwitch
---
```

```
*snip*
```

Example 28 **Interface connectivity resource**

6.3.12 Fabric creation (underlay and overlay)

The following manifest file demonstrates how a collapsed spine EVPN VXLAN fabric is orchestrated via EDA by using IPv6 link-local addressing and enabling MP-BGP peering (eBGP) between the collapsed spine nodes, carrying multiple address families. Several inputs are provided into the manifest file, which includes the IP pool for system0 assignment, ASN pool for the spines, label selectors for the spines, and the interswitch links between the two collapsed spine nodes.

The point-to-point interfaces between the collapsed spine nodes are also enabled for BFD, with BGP configured for fast failover.

```
# fabric deployment for collapsed spine design
apiVersion: fabrics.eda.nokia.com/v1alpha1
kind: Fabric
metadata:
  name: dc1-collapsed-spine
  namespace: 2-way-collapsed-spine
spec:
  systemPoolIPV4: system0
  leafs:
    leafNodeSelector:
      - eda.nokia.com/role=collapsed-spine
    asnPool: collapsed-spine-asn
  interSwitchLinks:
    unnumbered: IPV6
    linkSelector:
      - eda.nokia.com/role=interSwitch
  underlayProtocol:
    protocol:
      - EBGP
    bfd:
      enabled: true
      detectionMultiplier: 3
      minEchoReceiveInterval: 250000
      desiredMinTransmitInt: 250000
      requiredMinReceive: 250000
    bgp:
      asnPool: collapsed-spine-asn
  overlayProtocol:
    protocol: EBGP
```

Example 29 **Fabric deployment in EDA**

6.3.13 Bridge domain creation

Bridge domains are instantiated as MAC VRFs on SR Linux nodes. In this collapsed spine design, two kinds of bridge domains are created in EDA: EVPN VXLAN BDs (for the collapsed spine nodes) and simple BDs (for the ToR switches). The EVPN VXLAN BDs (MAC VRFs) map to a VXLAN VNI and an EVPN instance (EVI), which enables them for EVPN learning.

```
# EVPN VXLAN BDs
apiVersion: services.eda.nokia.com/v1alpha1
kind: BridgeDomain
metadata:
  name: macvrf-v10
  namespace: 2-way-collapsed-spine
spec:
  type: EVPNVXLAN
  vni: 10010
  evi: 10
  tunnelIndexPool: tunnel-index-pool
  macAging: 300
  macDuplicationDetection:
    enabled: true
    holdDownTime: 9
    monitoringWindow: 3
    action: StopLearning
    numMoves: 5
---
apiVersion: services.eda.nokia.com/v1alpha1
kind: BridgeDomain
metadata:
  name: macvrf-v20
  namespace: 2-way-collapsed-spine
spec:
  type: EVPNVXLAN
  vni: 10020
  evi: 20
  tunnelIndexPool: tunnel-index-pool
  macAging: 300
  macDuplicationDetection:
    enabled: true
    holdDownTime: 9
    monitoringWindow: 3
    action: StopLearning
    numMoves: 5
---
# Simple BDs for ToRs
apiVersion: services.eda.nokia.com/v1alpha1
```

```
kind: BridgeDomain
metadata:
  name: v10-simple
  namespace: 2-way-collapsed-spine
spec:
  type: SIMPLE
  macAging: 300
  macDuplicationDetection:
    enabled: true
    holdDownTime: 9
    monitoringWindow: 3
    action: StopLearning
    numMoves: 5
---
apiVersion: services.eda.nokia.com/v1alpha1
kind: BridgeDomain
metadata:
  name: v50-simple
  namespace: 2-way-collapsed-spine
spec:
  type: SIMPLE
  macAging: 300
  macDuplicationDetection:
    enabled: true
    holdDownTime: 9
    monitoringWindow: 3
    action: StopLearning
    numMoves: 5
---

*snip*
```

Example 30 **MAC-VRFs, VNIs and EVIs**

6.3.14 IRB interfaces

IRB interfaces, when deployed within the fabric, facilitate routing between Layer 2 VNIs in an EVPN VXLAN deployment. The following manifest file demonstrates how IRB interfaces are created in EDA. In the case of a VLAN (bridge domain) that is Layer 2-stretched across the fabric, the Layer 3 proxy-ARP functionality should be enabled for the respective IRB sub interface.

```
# IRB interfaces
apiVersion: services.eda.nokia.com/v1alpha1
kind: IRBInterface
metadata:
  name: irb-v10
```

```
  namespace: 2-way-collapsed-spine
spec:
  bridgeDomain: macvrf-v10
  router: vrf1
  ipMTU: 9200
  learnUnsolicited: BOTH
  ipAddresses:
    - ipv4Address:
        ipPrefix: 172.16.10.254/24
        primary: true
    - ipv6Address:
        ipPrefix: 2001:db8:0:10::254/64
        primary: true
  arpTimeout: 250
  evpnRouteAdvertisementType:
    arpDynamic: true
    ndDynamic: true
  hostRoutePopulate:
    dynamic: false
    evpn: false
    static: false
  l3ProxyARPND:
    proxyARP: true
    proxyND: true
---
apiVersion: services.eda.nokia.com/v1alpha1
kind: IRBInterface
metadata:
  name: irb-v20
  namespace: 2-way-collapsed-spine
spec:
  bridgeDomain: macvrf-v20
  router: vrf1
  ipMTU: 9200
  learnUnsolicited: BOTH
  ipAddresses:
    - ipv4Address:
        ipPrefix: 172.16.20.254/24
        primary: true
    - ipv6Address:
        ipPrefix: 2001:db8:0:20::254/64
        primary: true
  arpTimeout: 14400
  evpnRouteAdvertisementType:
    arpDynamic: true
    ndDynamic: true
  hostRoutePopulate:
```

```
    dynamic: false
    evpn: false
    static: false
  l3ProxyARPND:
    proxyARP: true
    proxyND: true
---
```

Example 31 **IRB interface manifest**

6.3.15 IP VRF creation

The following manifest file demonstrates how IP VRFs are created in EDA. This includes a Layer 3 VNI (which has a 1:1 mapping to the IP VRF) and an EVI. As these custom resources do not include any label selector, the VRF creation is determined based on any BDs that reference the IP VRF.

```
# VRFs (Routers in EDA)
# deployed without nodeSelector, gets pushed based on BD reference
apiVersion: services.eda.nokia.com/v1alpha1
kind: Router
metadata:
  name: vrf1
  namespace: 2-way-collapsed-spine
spec:
  type: EVPNVXLAN
  vni: 10500
  evi: 500
  tunnelIndexPool: tunnel-index-pool
---
```

Example 32 **VRF creation**

6.3.16 VLAN creation

The following manifest file demonstrates how VLANs are created in EDA, using examples of an untagged and tagged Layer 2 deployment. The label selectors determine which interfaces the VLANs are deployed on.

```
# VLANs untagged for EVPN VXLAN BDs
apiVersion: services.eda.nokia.com/v1alpha1
kind: VLAN
metadata:
  name: untagged-v10
  namespace: 2-way-collapsed-spine
spec:
```

```
  bridgeDomain: macvrf-v10
  interfaceSelector:
    - eda.nokia.com/untagged-v10=enabled
  vlanID: untagged
---
# VLANs untagged for SIMPLE BDs
apiVersion: services.eda.nokia.com/v1alpha1
kind: VLAN
metadata:
  name: untagged-v10-tor
  namespace: 2-way-collapsed-spine
spec:
  bridgeDomain: v10-simple
  interfaceSelector:
    - eda.nokia.com/untagged-v10-simple=enabled
  vlanID: untagged
---
# VLANs tagged for EVPN VXLAN BDs
apiVersion: services.eda.nokia.com/v1alpha1
kind: VLAN
metadata:
  name: tagged-v10
  namespace: 2-way-collapsed-spine
spec:
  bridgeDomain: macvrf-v10
  interfaceSelector:
    - eda.nokia.com/tagged-v10=enabled
  vlanID: '10'
---
```

Example 33 **VLAN creation**

6.3.17 Routed interfaces in EDA

Routed interfaces are created in EDA using the following manifest:

```
# routed interface on spine2
apiVersion: services.eda.nokia.com/v1alpha1
kind: RoutedInterface
metadata:
  name: spine2-ixia-1
  namespace: 2-way-collapsed-spine
spec:
  arpTimeout: 14400
  interface: spine2-ixia-1
  ipMTU: 9200
```

```
  ipv4Addresses:
    - ipPrefix: 172.16.100.0/31
      primary: true
  ipv6Addresses:
    - ipPrefix: 2001:db8:0:100::0/127
      primary: true
  router: vrf1
  vlanID: "100"
---
```

Example 34 **Routed interface in EDA**

6.3.18 Namespace in EDA

EDA allows for logical separation of resources in their own namespaces, as shown below. All resources specific to this namespace must reference this namespace in their manifest files.

```
# namespace for collapsed spine
apiVersion: core.eda.nokia.com/v1
kind: Namespace
metadata:
  name: 2-way-collapsed-spine
  namespace: "eda-system"
spec:
  description: 'namespace for 2-spine collapsed NVD'
---
```

Example 35 **Namespaces in EDA**

6.3.19 EDA configlets

EDA provides the flexibility to input user-defined configuration (which may not be auto-generated by EDA in a particular version). This functionality is achieved via configlets. The collapsed spine validated design uses configlets to:

- enable the advertisement of ARP/ND entries only when corresponding MAC entries exist for a MAC VRF
- configure system-wide default MTUs
- enable appropriate logging for different subsystems in SR Linux

The configlets for advertising ARP/ND entries only when corresponding MAC entries exist is shown below (this must be done for all MAC VRFs, with only one example shown below):

```
apiVersion: config.eda.nokia.com/v1alpha1
kind: Configlet
metadata:
  name: macvrf-v10-arp-nd
  namespace: 2-way-collapsed-spine
spec:
  endpointSelector:
    - eda.nokia.com/role=collapsed-spine
  operatingSystem: srl
  priority: 100
  configs:
    - path: .network-instance{.name=="macvrf-v10"}.protocols.bgp-evpn.bgp-instance{.id==1}
      operation: Create
      config: |-
        {
          "routes": {
            "bridge-table": {
              "mac-ip": {
                "advertise-arp-nd-only-with-mac-table-entry": true
              }
            }
          }
        }

*snip*
```

Example 36 **Configlet for ARP/ND advertisement**

The configlets for system-wide MTU settings is shown below:

```
apiVersion: config.eda.nokia.com/v1alpha1
kind: Configlet
metadata:
  name: system-wide-mtu
  namespace: 2-way-collapsed-spine
spec:
  endpointSelector:
    - eda.nokia.com/role=collapsed-spine
    - eda.nokia.com/role=tor
  operatingSystem: srl
  priority: 100
  configs:
    - path: .system.mtu
      operation: Create
      config: |-
        {
          "default-ip-mtu": "9200",
```

```
      "default-l2-mtu": "9412",
      "default-port-mtu": "9412"
    }
```

Example 37 **Configlet for system-wide MTU settings**

The configlets for system logging of different subsystems in SR Linux is shown below:

```
apiVersion: config.eda.nokia.com/v1alpha1
kind: Configlet
metadata:
  name: system-logging-spines
  namespace: 2-way-collapsed-spine
spec:
  endpointSelector:
    - eda.nokia.com/role=collapsed-spine
  operatingSystem: srl
  priority: 100
  configs:
    - path: .system.logging
      operation: Create
      config: |-
        {
          "buffer": [
            {
              "buffer-name": "log-buffer",
              "subsystem": [
                {
                  "subsystem-name": "arpnd",
                  "priority": {
                    "match-above": "informational"
                  }
                },
                {
                  "subsystem-name": "bfd",
                  "priority": {
                    "match-above": "informational"
                  }
                },
                {
                  "subsystem-name": "bgp",
                  "priority": {
                    "match-above": "informational"
                  }
                },
                {
                  "subsystem-name": "evpn",
```

```
            "priority": {
              "match-above": "informational"
            }
          },
          {
            "subsystem-name": "lag",
            "priority": {
              "match-above": "informational"
            }
          },
          {
            "subsystem-name": "lldp",
            "priority": {
              "match-above": "informational"
            }
          },
          {
            "subsystem-name": "vxlan",
            "priority": {
              "match-above": "informational"
            }
          }
        ]
      }
    ]
  }
---
```

Example 38 **Configlet for system logging**

6.3.20 Custom topology view for collapsed spine design

To accurately view the collapsed spine design in the topology viewer, the topology manifest can be updated to include the labels for the ToR switches, as shown below.

```
apiVersion: topologies.eda.nokia.com/v1alpha1
kind: TopologyGrouping
metadata:
  name: role # no namespace-specific topology can be configured; it must be the same role topology in eda-system ns
  namespace: eda-system
spec:
  tierSelectors:
  - nodeSelector:
    - eda.nokia.com/role=backbone
    tier: 1
  - nodeSelector:
```

```
  - eda.nokia.com/role=superspine
  tier: 2
- nodeSelector:
  - eda.nokia.com/role=spine
  tier: 3
- nodeSelector:
  - eda.nokia.com/role=collapsed-spine
  tier: 3
- nodeSelector:
  - eda.nokia.com/role=borderleaf
  tier: 4
- nodeSelector:
  - eda.nokia.com/role=leaf
  tier: 4
- nodeSelector:
  - eda.nokia.com/role=tor
  tier: 4
- nodeSelector:
  - eda.nokia.com/role=rail
  tier: 4
- tier: 4
uiDescription: Assign nodes to tiers based on eda.nokia.com/role labels.
uiName: Role
```

Example 39 **Topology manifest showing a collapsed spine design**

6.4 EDA workflows via UI

6.4.1 Node profiles for node onboarding

Node profiles are specified during node onboarding and are used to determine the IP pool from which to assign an IP address to the node, what the gNMI discovery port is, and the username/password credentials to log into the device. Node profiles can be created by navigating to **Main → Node Profiles**.

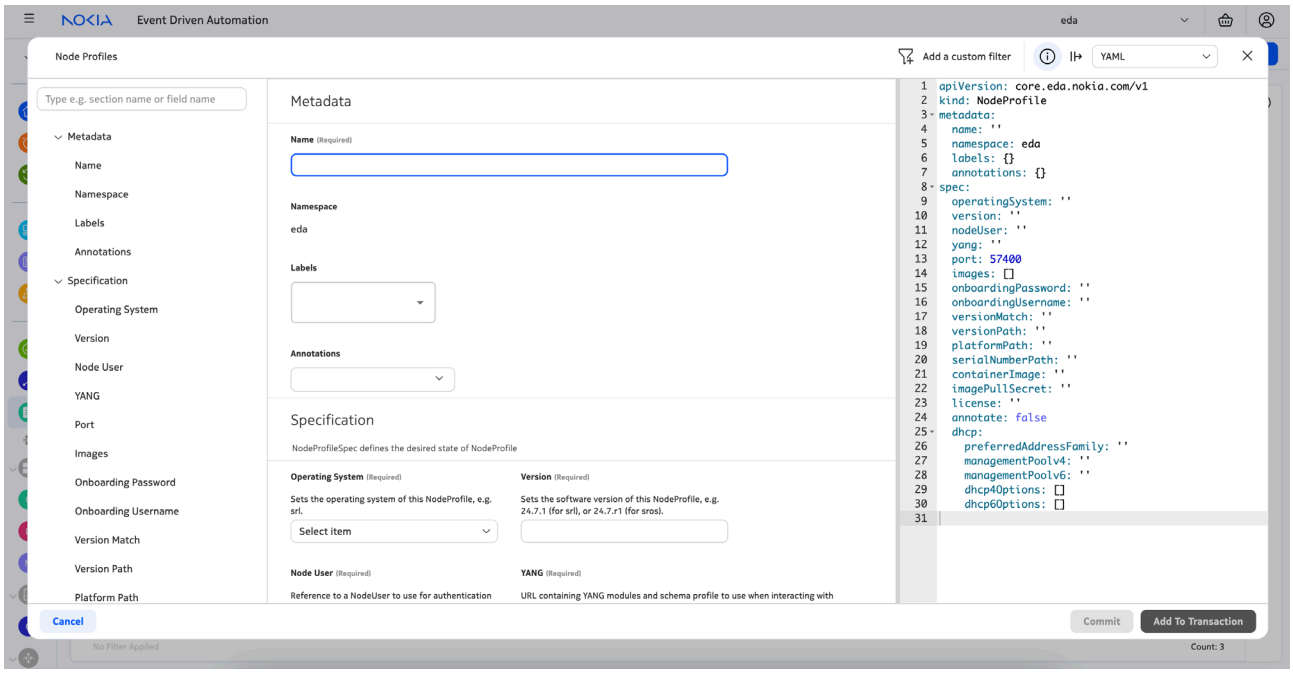

Figure 18 **Node profile creation page in EDA UI**

6.4.2 ASN pools for leafs and spines

The ASN pools are created as indices pools, which can then be assigned to leafs and spines during fabric creation. These indices pools can be viewed and created by navigating to **Main** → **Indices**.

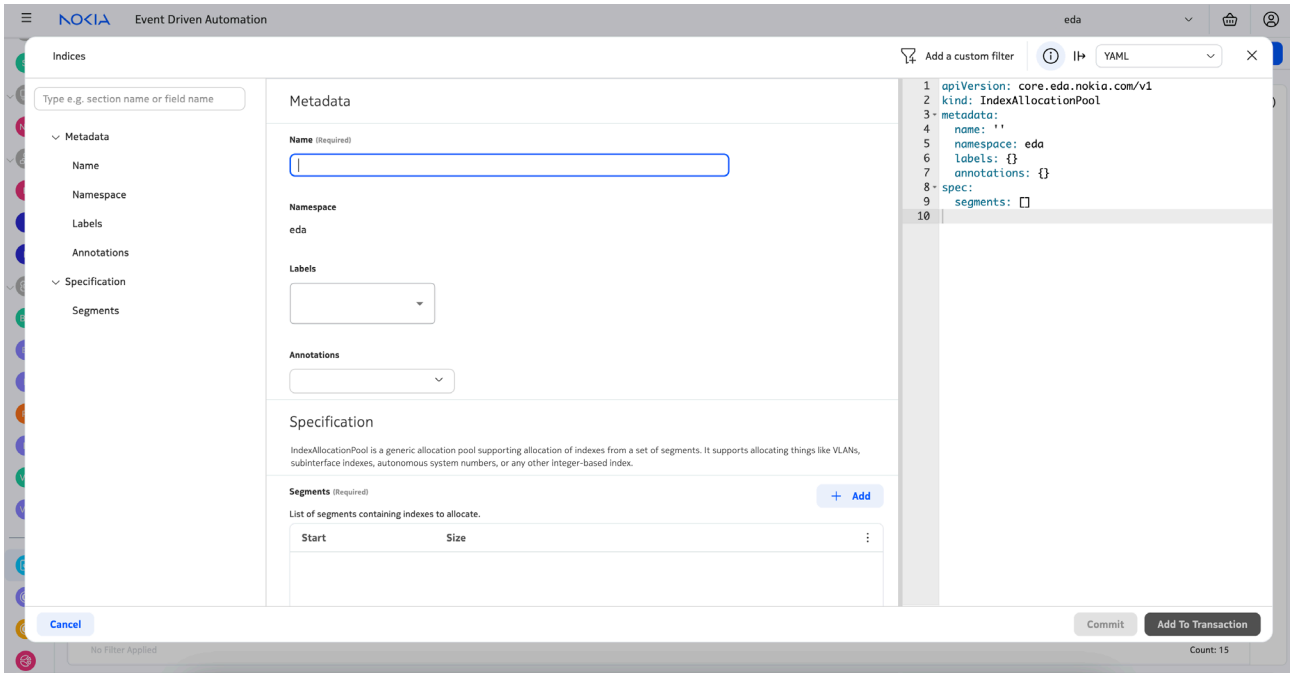

Figure 19 **ASN creation as an indices pool in EDA UI**

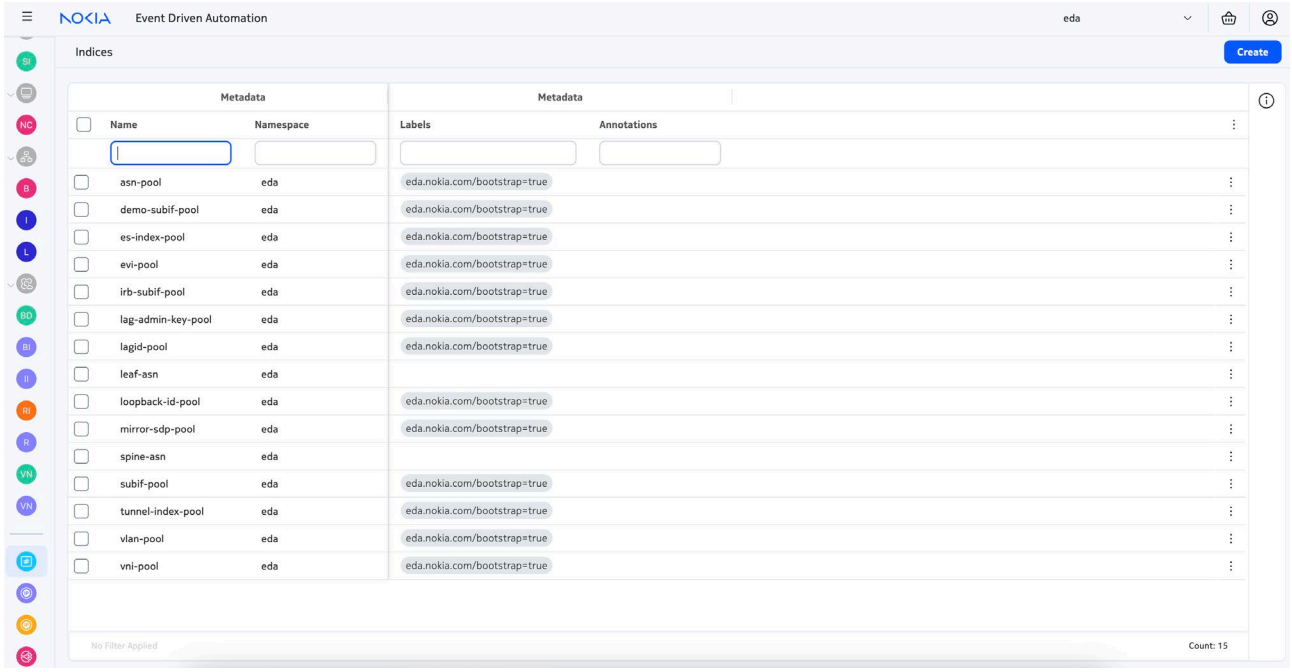

Figure 20 **List of all indices pools (default and user-defined) in EDA UI**

6.4.3 IP pool creation allocation

IP pools can be created for multiple reasons, for example, a subnet allocation or an exact IP address allocation. In the case of this NVD, an IP pool of type IP Addresses is created to assign a unique IPv4 address from an IPv4 subnet for the system0 interface of nodes in the fabric. This type of IP pool can be created by navigating to **Main** → **IP Addresses**.

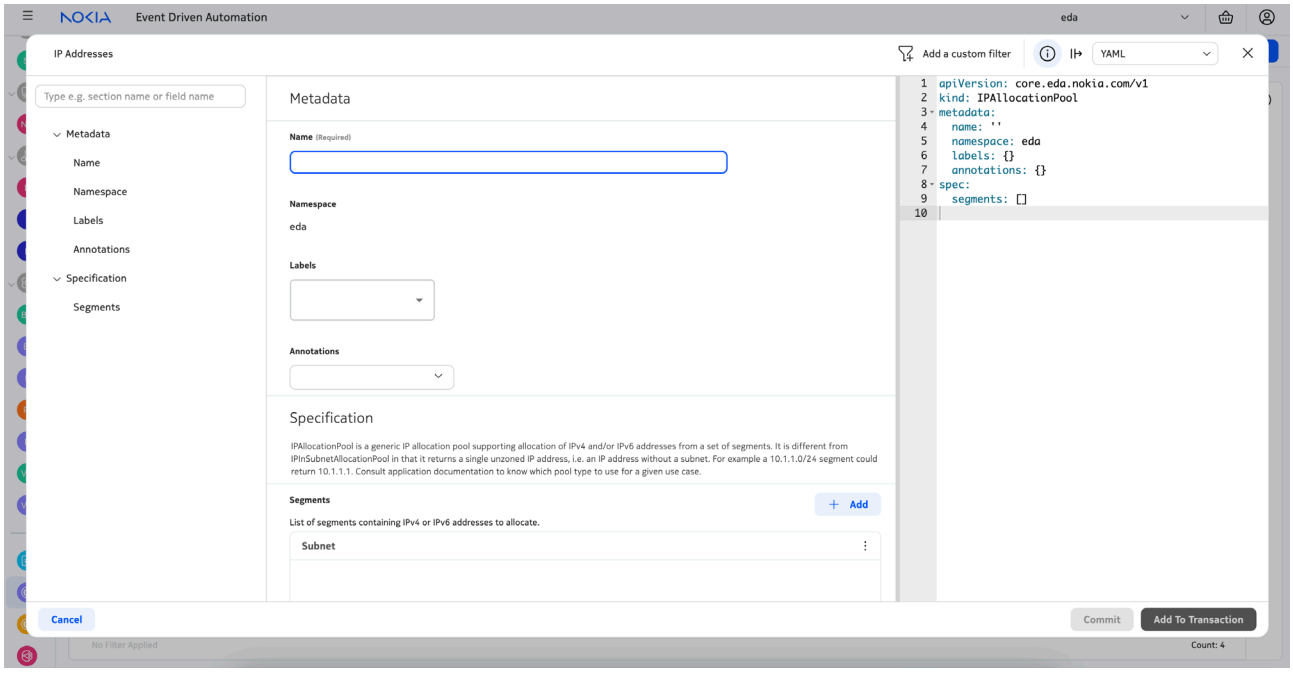

Figure 21 **IP pool creation in EDA UI**

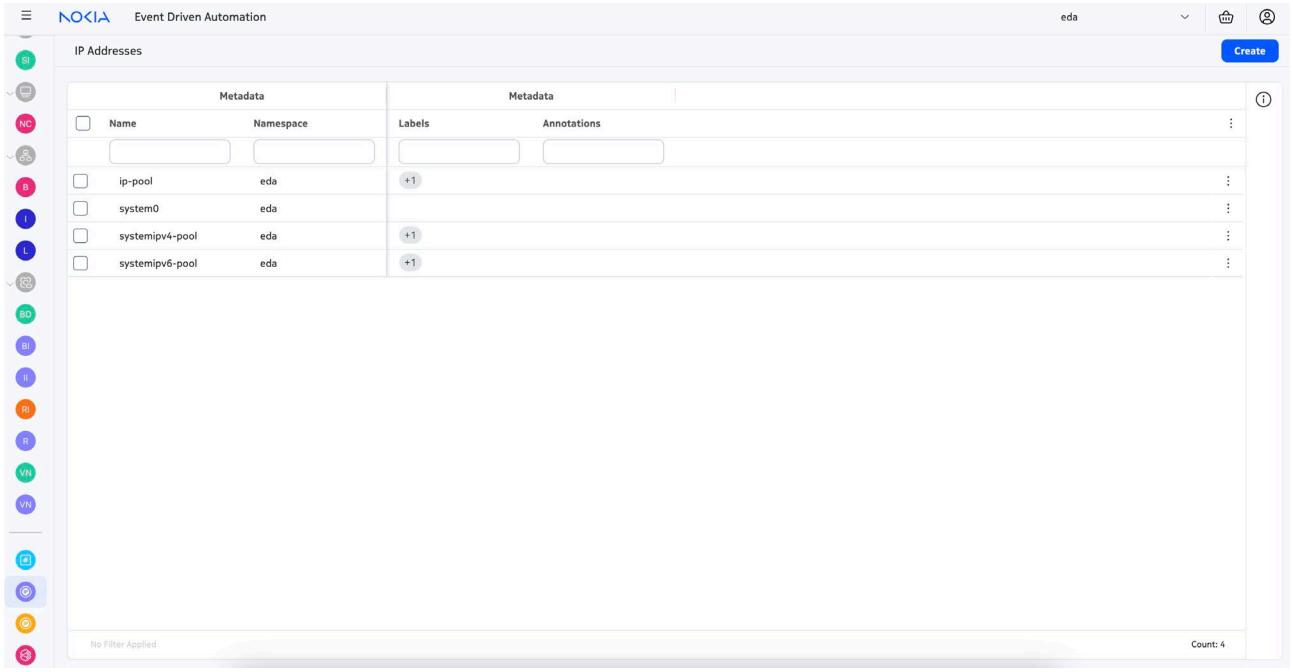

Figure 22 **List of all IP pools (default and user-defined) in EDA UI**

6.4.4 Onboarding nodes

Nodes can be created and viewed by navigating to **Main** → **Nodes**. These are nodes onboarded into the fabric and represented in the topology view.

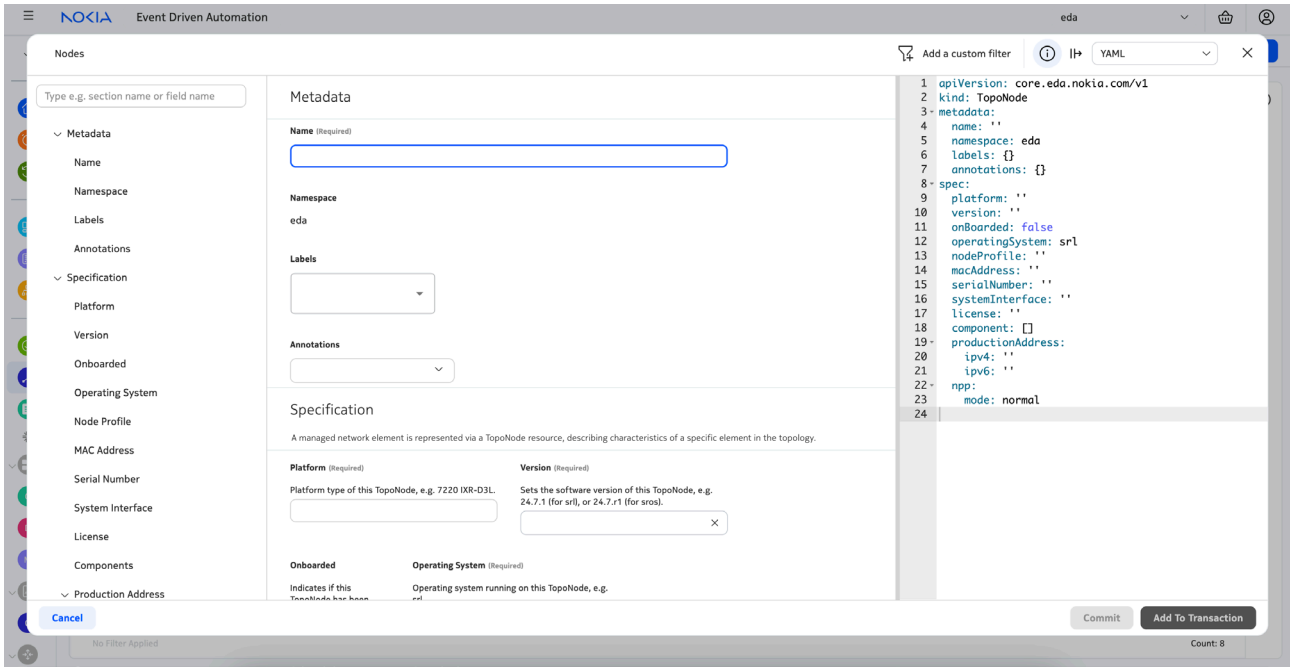

Figure 23 **Node creation (onboarding) in EDA UI**

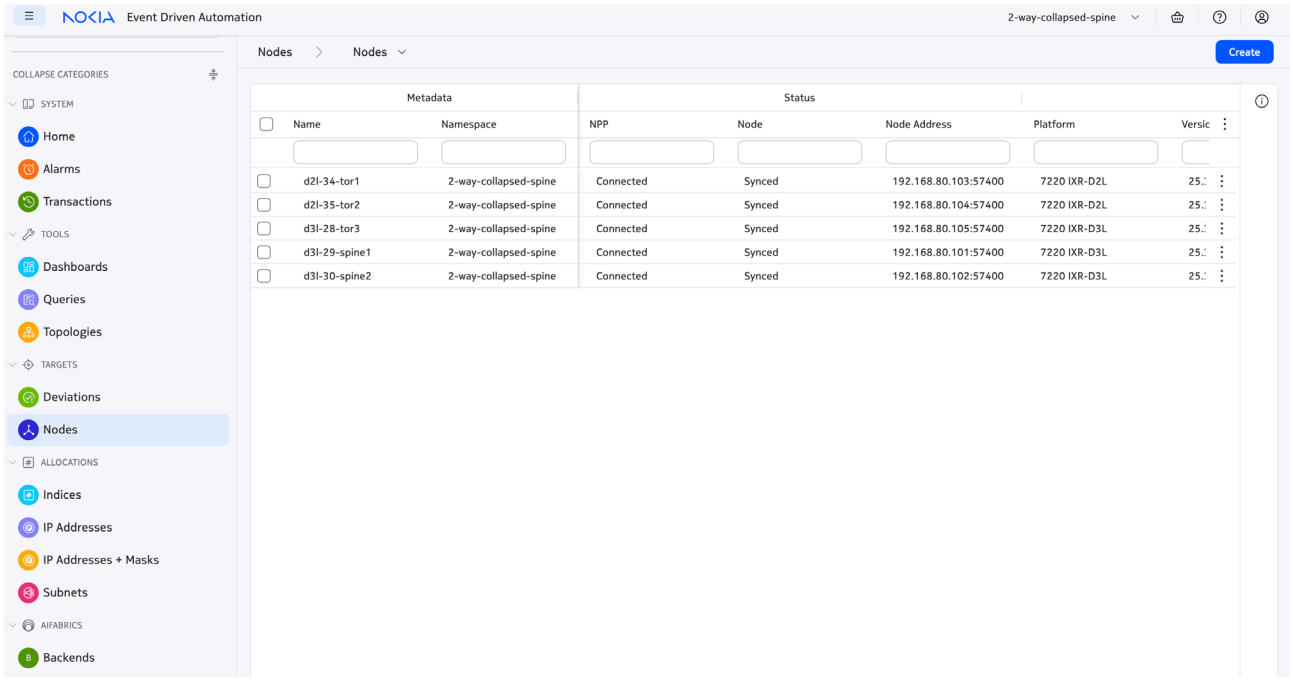

Figure 24 **List of all onboarded nodes (along with monitored parameters) in EDA UI**

6.4.5 Fabric creation

Fabrics can be created by navigating to **Main** → **Fabrics**. This instantiates all fabric nodes (based on label selector) and pushes the generated fabric configuration per-node.

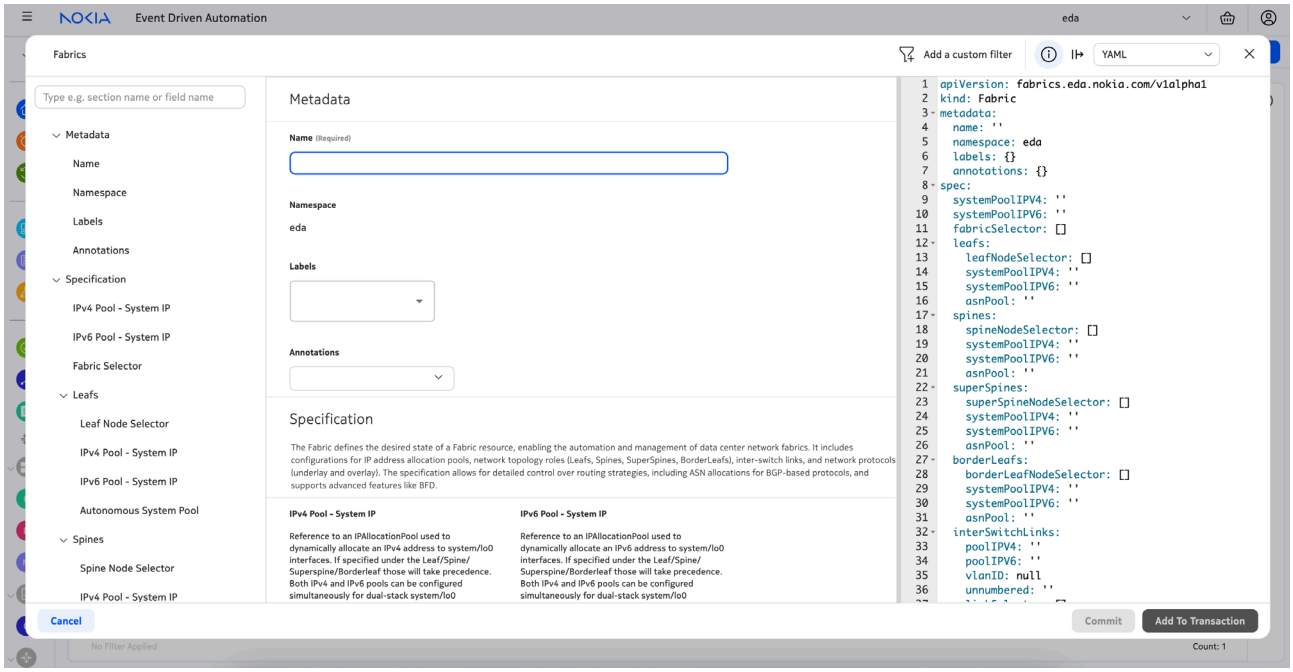

Figure 25 **Fabric creation in EDA UI**

6.4.6 Bridge domains

Bridge domains are instantiated as MAC VRFs on fabric nodes and can be created by navigating to **Main** → **Virtual Networks** → **Bridge Domains**.

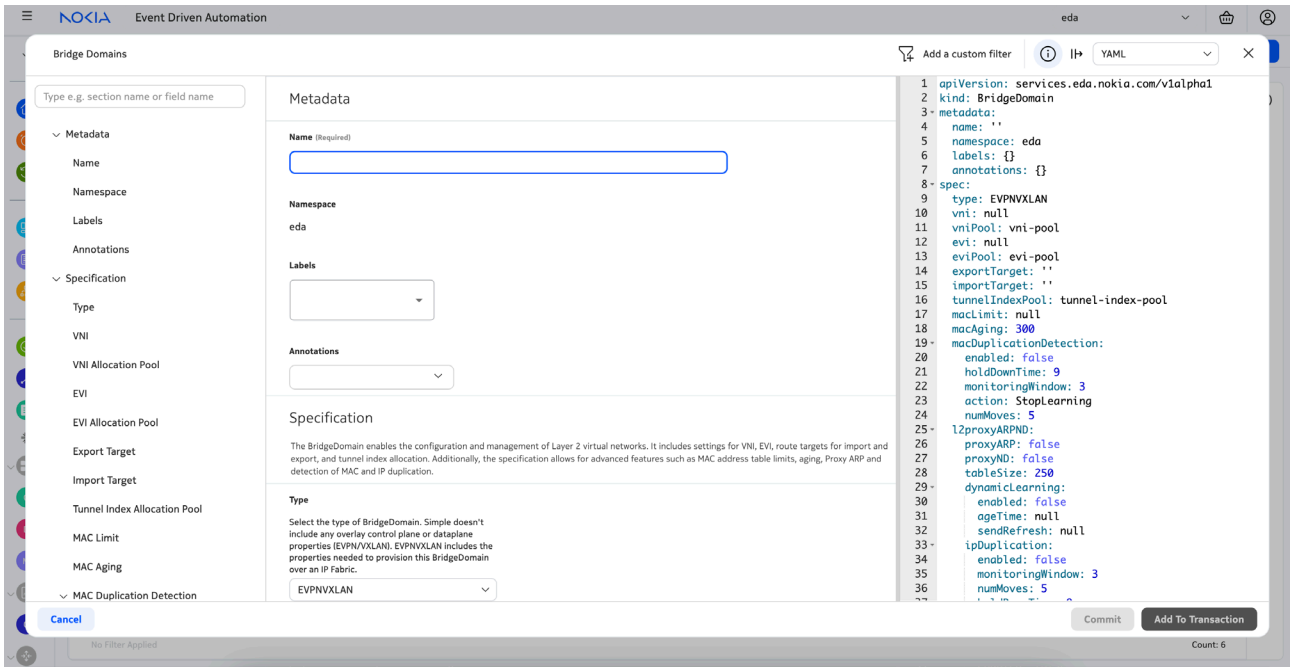

Figure 26 **Bridge domain creation in EDA UI**

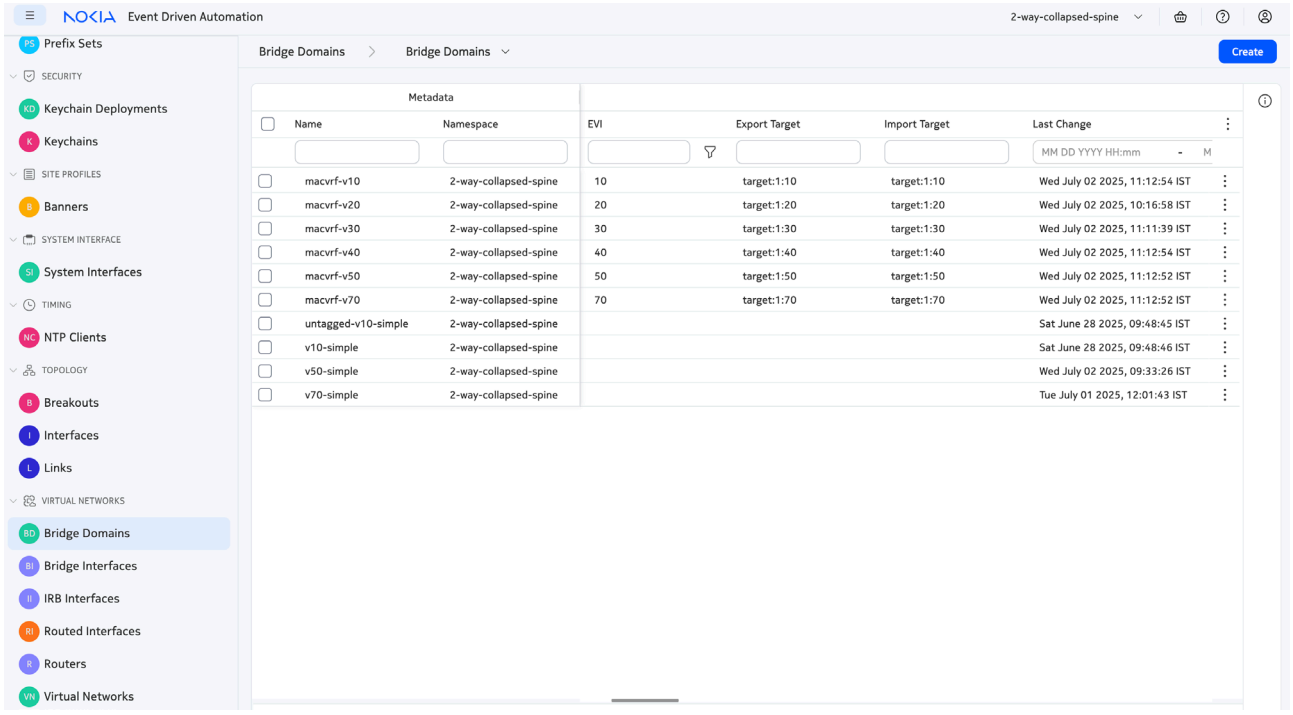

Figure 27 **List of all bridge domains in EDA UI**

6.4.7 IRB interfaces

IRB interfaces act as the default gateways for services connected to the leafs and are deployed using an anycast, distributed gateway model. IRB interfaces can be created by navigating to **Main → Virtual Networks → IRB Interfaces**.

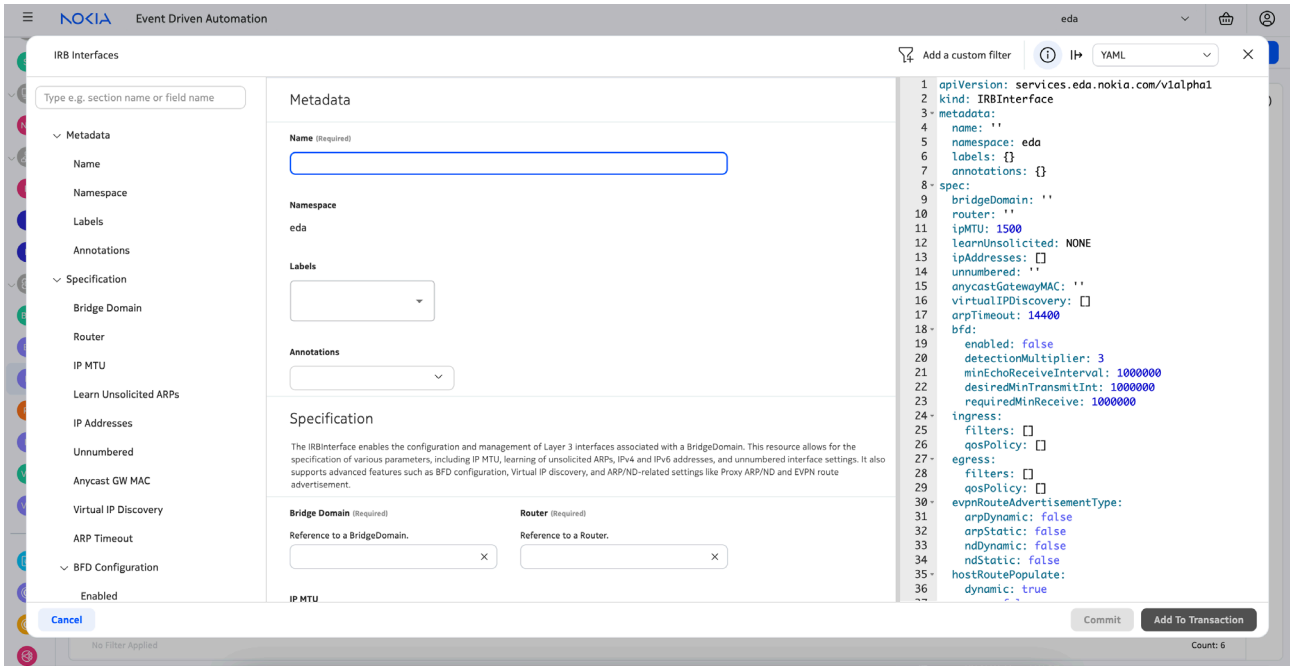

Figure 28 **IRB interface creation in EDA UI**

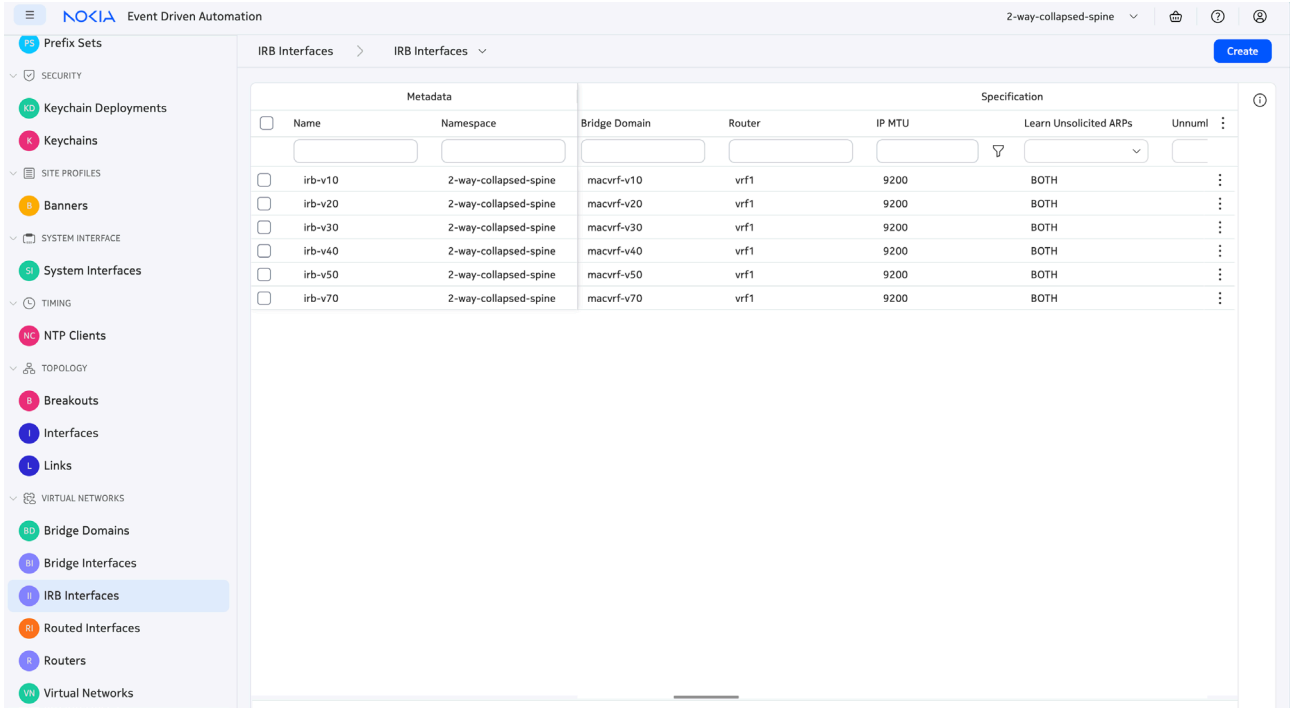

Figure 29 **List of all IRB interfaces in EDA UI**

6.4.8 IP VRFs (Routers)

IP VRFs are used to provide multitenancy and Layer 3 isolation. IP VRFs can be created by navigating to **Main → Virtual Networks → Routers**.

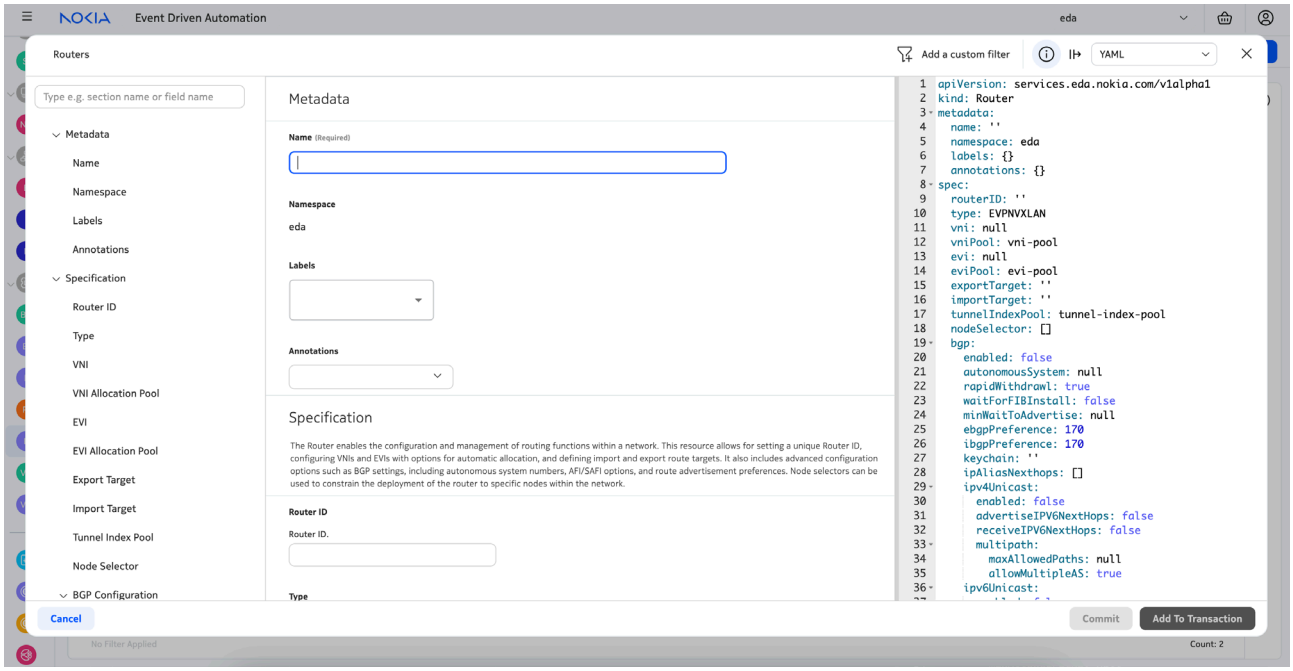

Figure 30 **IP VRF (router) creation in EDA UI**

6.4.9 VLANs

VLANs can be created by navigating to **Main** → **Virtual Networks** → **VLANs**.

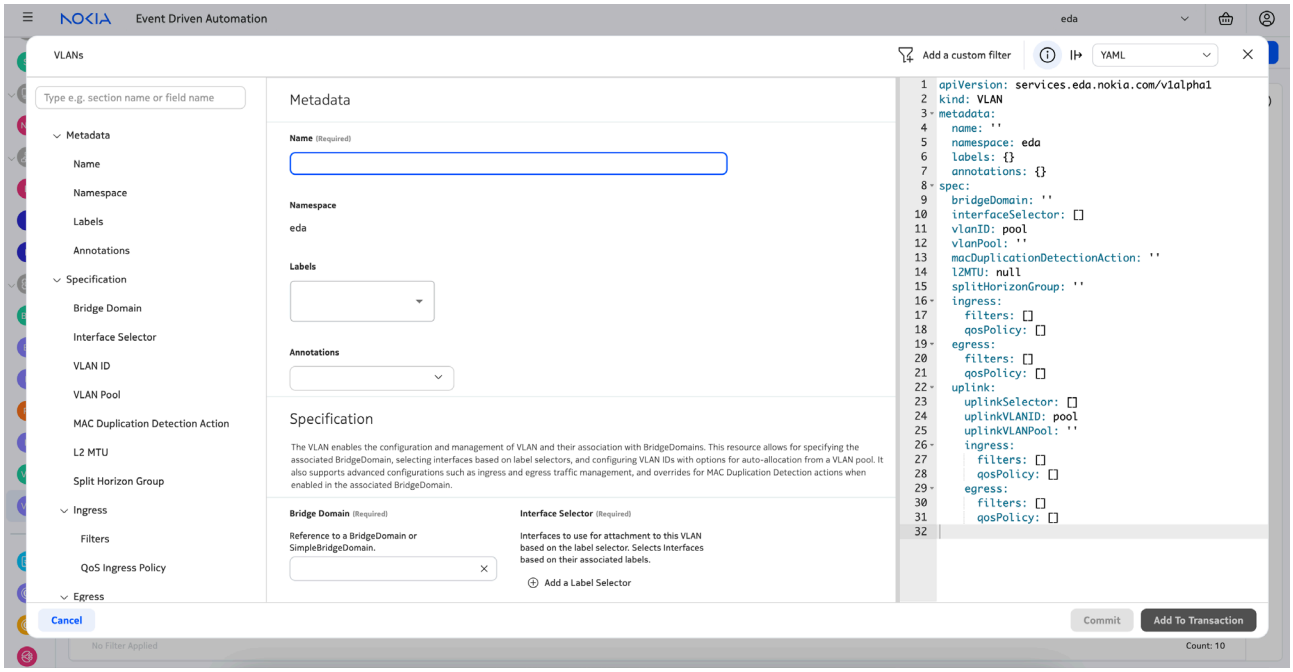

Figure 31 **VLAN creation in EDA UI**

6.4.10 Configlets for custom configuration

Configlets allow for supplemental configuration that can be added to the per-node configuration generated by EDA. Configlets can be created by navigating to **Main** → **Configuration** → **Configlets**.

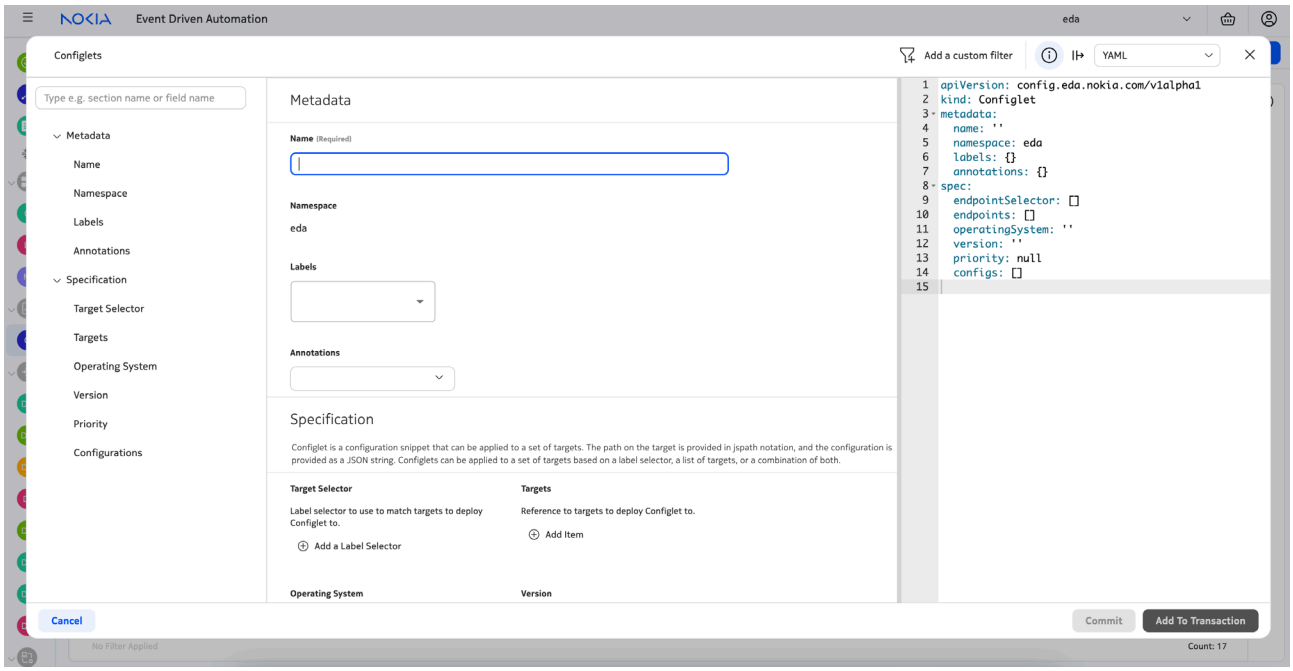

Figure 32 **Configuration configlets creation in EDA UI**

7 Validation

7.1 Network validation

7.1.1 Underlay and overlay

The underlay comprises point-to-point IPv6 link-local addressing with IPv6 neighbor discovery to discover the peer on its local link. BGP dynamic discovery is then used to establish MP-BGP peering with the neighbor, with IPv4, IPv6, and EVPN address families (and sub-address families) exchanged as capabilities over this single peering.

The discovered neighbors on an IPv6 interface can be confirmed using *info from state interface [interface] subinterface [subinterface] ipv6 neighbor-discovery*.

```
A:admin@d31-29-spine1# info from state interface ethernet-1/31 subinterface 0 ipv6
neighbor-discovery
    duplicate-address-detection true
    reachable-time 30
    stale-time 14400
    learn-unsolicited none
    proxy-nd false
    neighbor fe80::eaf3:75ff:fe05:52ad {
        link-layer-address E8:F3:75:05:52:AD
        origin dynamic
        is-router true
        current-state stale
```

```
        next-state-time "2025-07-05T05:45:22.609Z (an hour from now)"
        datapath-programming {
            status success
        }
    }
    limit {
        log-only false
        warning-threshold-pct 90
    }
```

Example 40 **Interface discovery**

The state of BGP neighbors can be confirmed using show network-instance default protocols bgp neighbor, assuming that the BGP peers are configured for the default network-instance.

```
A:admin@d3l-29-spine1# show network-instance default protocols bgp neighbor  | as yaml
---
neighbor summary:
  - network-instance: default
    state:
      - Net-Inst: default
      - Peer: 'fe80::eaf3:75ff:fe05:52ad%ethernet-1/31.0'
        Group: bgpgroup-ebgp-dc1-collapsed-spine
        Flags: DB
        Peer-AS: 65502
        State: established
        Uptime: '2d:22h:40m:59s'
        AFI/SAFI: evpn\nipv4-unicast\nipv6-unicast
        '[Rx/Active/Tx]': '[64/53/66]\n[1/1/1]\n[0/0/0]'
      - Net-Inst: default
      - Peer: 'fe80::eaf3:75ff:fe05:52ae%ethernet-1/32.0'
        Group: bgpgroup-ebgp-dc1-collapsed-spine
        Flags: DB
        Peer-AS: 65502
        State: established
        Uptime: '2d:22h:40m:59s'
        AFI/SAFI: evpn\nipv4-unicast\nipv6-unicast
        '[Rx/Active/Tx]': '[130/0/130]\n[2/1/2]\n[0/0/0]'
    config:
      - network-instance: default
        configured-peers: 0
        configured-up-peers: 0
        disabled-peers: 0
        dynamic-peers: 2
```

Example 41 **BGP neighborship**

BFD is used for fast-failover in the NVD. The BFD session state can be confirmed using *info from state bfd network-instance default peer [index].*

```
A:admin@d3l-29-spine1# info from state bfd network-instance default peer *
    peer 16387 {
        oper-state up
        ipv6-link-local-interface ethernet-1/31.0
        local-address fe80::eaf3:75ff:fe05:3aad
        remote-address fe80::eaf3:75ff:fe05:52ad
        remote-discriminator 16385
```

```
        subscribed-protocols BGP
        session-state UP
        remote-session-state UP
        last-state-transition "2025-07-02T05:36:35.998Z (2 days ago)"
        failure-transitions 0
        local-diagnostic-code NO_DIAGNOSTIC
        remote-diagnostic-code NO_DIAGNOSTIC
        remote-minimum-receive-interval 250000
        remote-control-plane-independent false
        active-transmit-interval 250000
        active-receive-interval 250000
        remote-multiplier 3
        async {
            last-packet-transmitted "2025-07-05T04:18:03.357Z (3 seconds ago)"
            last-packet-received "2025-07-05T04:18:03.228Z (3 seconds ago)"
            transmitted-packets 1284774
            received-packets 1284831
            up-transitions 1
        }
    }
    peer 16388 {
        oper-state up
        ipv6-link-local-interface ethernet-1/32.0
        local-address fe80::eaf3:75ff:fe05:3aae
        remote-address fe80::eaf3:75ff:fe05:52ae
        remote-discriminator 16386
        subscribed-protocols BGP
        session-state UP
        remote-session-state UP
        last-state-transition "2025-07-02T05:36:36.203Z (2 days ago)"
        failure-transitions 0
        local-diagnostic-code NO_DIAGNOSTIC
        remote-diagnostic-code NO_DIAGNOSTIC
        remote-minimum-receive-interval 250000
        remote-control-plane-independent false
        active-transmit-interval 250000
        active-receive-interval 250000
        remote-multiplier 3
        async {
            last-packet-transmitted "2025-07-05T04:18:03.399Z (2 seconds ago)"
            last-packet-received "2025-07-05T04:18:03.399Z (2 seconds ago)"
            transmitted-packets 1284805
            received-packets 1284803
            up-transitions 1
        }
```

Example 42 **BFD information**

In addition to viewing routes in BGP RIB-In using *show network-instance default protocols bgp routes [family] summary*, all routes advertised and received via BGP can also be confirmed using the commands *show network-instance default protocols bgp neighbor [neighbor] advertising-routes [family]* and *show network-instance default protocols bgp neighbor [neighbor] received-routes [family]*.

```
A:admin@d31-29-spine1# show network-instance default protocols bgp routes ipv4 summary |
as yaml
---
header:
  - Header: default
```

```
    net-inst: default
    routes:
      - Status: u*>
      - Network: 192.0.2.1/32
      - Next Hop: 'fe80::eaf3:75ff:fe05:52ad%ethernet-1/31.0'
        LocPref: 100
        Path Val: ' i[65502]'
      - Status: u*>
      - Network: 192.0.2.1/32
      - Next Hop: 'fe80::eaf3:75ff:fe05:52ae%ethernet-1/32.0'
        LocPref: 100
        Path Val: ' i[65502]'
      - Status: u*>
      - Network: 192.0.2.2/32
      - Next Hop: 0.0.0.0
        Path Val:  i
      - Status:
      - Network: 192.0.2.2/32
      - Next Hop: 'fe80::eaf3:75ff:fe05:52ae%ethernet-1/32.0'
        LocPref: 100
        Path Val: ' i[65502, 65501]'
    stats:
      - network instance: default
        routes received: 4
        used count: 3
        stale count: 0
        valid count: 3
        available dest: 3
        ecmp multipath count: 1
```

Example 43 **BGP routes**

7.1.2 Link aggregation

Interface state (and overall LAG state) can be viewed using *show lag [lag-interface] brief* or *show lag [lag-interface] lacp-state* for LACP-enabled LAGs.

```
A:admin@d3l-29-spine1# show lag lacp-state | as yaml
---
LacpHeader:
  - Lag Id: lag1
    LacpBrief:
      Interval: FAST
      Mode: ACTIVE
      System Id: '00:00:00:00:00:12'
      System Priority: 32768
    LacpState:
      - Members: ethernet-1/25
        Oper state: up
        Activity: ACTIVE
        Timeout: SHORT
        State: IN_SYNC/True/True/True
        System Id: '00:00:00:00:00:12'
        Oper key: 5
        Partner Id: '00:00:11:11:11:11'
        Partner Key: 6
        Port No: 2
        Partner Port No: 1
```

```
  - Lag Id: lag2
    LacpBrief:
      Interval: FAST
      Mode: ACTIVE
      System Id: '00:00:00:00:12:00'
      System Priority: 32768
    LacpState:
      - Members: ethernet-1/26
        Oper state: up
        Activity: ACTIVE
        Timeout: SHORT
        State: IN_SYNC/True/True/True
        System Id: '00:00:00:00:12:00'
        Oper key: 1
        Partner Id: '00:00:22:22:22:22'
        Partner Key: 3
        Port No: 3
        Partner Port No: 1
  - Lag Id: lag3
    LacpBrief:
      Interval: FAST
      Mode: ACTIVE
      System Id: '00:00:00:12:00:00'
      System Priority: 32768
    LacpState:
      - Members: ethernet-1/29
        Oper state: up
        Activity: ACTIVE
        Timeout: SHORT
        State: IN_SYNC/True/True/True
        System Id: '00:00:00:12:00:00'
        Oper key: 4
        Partner Id: '00:00:33:33:33:33'
        Partner Key: 2
        Port No: 4
        Partner Port No: 1
  - Lag Id: lag4
    LacpBrief:
      Interval: FAST
      Mode: ACTIVE
      System Id: '00:00:12:00:00:00'
      System Priority: 32768
    LacpState:
      - Members: ethernet-1/23
        Oper state: up
        Activity: ACTIVE
        Timeout: SHORT
        State: IN_SYNC/True/True/True
        System Id: '00:00:12:00:00:00'
        Oper key: 7
        Partner Id: '00:00:00:00:00:01'
        Partner Key: 1
        Port No: 1
        Partner Port No: 1
```

Example 44 **LAG state information**

7.1.3 Ethernet segments

For Ethernet segments, DF and non-DF status can be determined on a per-VRF basis.

```
A:admin@d31-29-spine1# show system network-instance ethernet-segments | as yaml
---
Ethernet-Segment:
  - Name: spine1-spine2-tor1-lag
    Oper State: up
    Oper multi homing: all-active
    Info:
      ESI: '00:00:00:00:00:00:12:00:00:00'
      Alg: default
      Peers: '192.0.2.1, 192.0.2.2'
      Interface: lag1
      Next-hop: N/A
      evi: N/A
    NetworkInstance:
      - Name: macvrf-v50
        Candidates: '192.0.2.1 (DF), 192.0.2.2'
        Interface: lag1.50
  - Name: spine1-spine2-tor2-lag
    Oper State: up
    Oper multi homing: all-active
    Info:
      ESI: '00:00:00:00:00:12:00:00:00:00'
      Alg: default
      Peers: '192.0.2.1, 192.0.2.2'
      Interface: lag2
      Next-hop: N/A
      evi: N/A
    NetworkInstance:
      - Name: macvrf-v10
        Candidates: '192.0.2.1 (DF), 192.0.2.2'
        Interface: lag2.4096
  - Name: spine1-spine2-tor3-lag
    Oper State: up
    Oper multi homing: all-active
    Info:
      ESI: '00:00:00:00:12:00:00:00:00:00'
      Alg: default
      Peers: '192.0.2.1, 192.0.2.2'
      Interface: lag3
      Next-hop: N/A
      evi: N/A
    NetworkInstance:
      - Name: macvrf-v70
        Candidates: '192.0.2.1 (DF), 192.0.2.2'
        Interface: lag3.70
  - Name: spine1-spine2-tor4-lag
    Oper State: up
    Oper multi homing: single-active
    Info:
      ESI: '00:00:00:12:00:00:00:00:00:00'
      Alg: preference
      Peers: '192.0.2.1, 192.0.2.2'
      Interface: lag4
      Next-hop: N/A
      evi: N/A
    NetworkInstance:
      - Name: macvrf-v10
        Candidates: '192.0.2.1, 192.0.2.2 (DF)'
        Interface: lag4.4096
      - Name: macvrf-v40
        Candidates: '192.0.2.1, 192.0.2.2 (DF)'
```

```
        Interface: lag4.40
Statistics:
  ES Up: 4
  ES Down: 0
```

Example 45 **Ethernet segment description**

7.1.4 MAC VRFs and MAC address learning

The bridge table per MAC VRF can be viewed using the commands shown below.

```
A:admin@d3l-29-spine1# show network-instance macvrf-v10 bridge-table mac-table all | as
yaml
---
Network:
  - Name: macvrf-v10
    Mac table:
      - Address: '00:00:5E:00:01:01'
        Destination: irb-interface
        Dest Index: 0
        Type: irb-interface-anycast
        Active: true
        Aging: N/A
        Last Update: '2025-07-02T04:41:48.000Z'
      - Address: '00:12:01:00:00:01'
        Destination: ethernet-1/27.4096
        Dest Index: 3
        Type: learnt
        Active: true
        Aging: 271
        Last Update: '2025-07-02T04:41:55.000Z'
      - Address: '00:13:01:00:00:01'
        Destination: 'vxlan-interface:vxlan0.501 vtep:192.0.2.1 vni:10010'
        Dest Index: 5080458
        Type: evpn
        Active: true
        Aging: N/A
        Last Update: '2025-07-02T05:36:36.000Z'
      - Address: '00:15:01:00:00:01'
        Destination: lag4.4096
        Dest Index: 15
        Type: learnt
        Active: true
        Aging: 180
        Last Update: '2025-07-02T05:38:17.000Z'
      - Address: 'E8:F3:75:05:3A:CF'
        Destination: irb-interface
        Dest Index: 0
        Type: irb-interface
        Active: true
        Aging: N/A
        Last Update: '2025-07-02T04:41:48.000Z'
      - Address: 'E8:F3:75:05:52:CF'
        Destination: 'vxlan-interface:vxlan0.501 vtep:192.0.2.1 vni:10010'
        Dest Index: 5080458
        Type: evpn-static
        Active: true
        Aging: N/A
        Last Update: '2025-07-02T05:36:36.000Z'
```

```
*snip*
```

Example 46 **Bridge table per MAC VRF**

7.1.5 Route validation in default network-instance and IP VRFs

The following command shows routes in the default network-instance. The default network-instance can be changed to user-defined network instances based on VRFs in use.

```
A:admin@d3l-29-spine1# show network-instance default route-table ipv4-unicast route
192.0.2.1 | as yaml
---
instance:
  - Name: default
    ip route:
      - Prefix: 192.0.2.1/32
      - ID: 0
      - Route Type: bgp
      - Route Owner: bgp_mgr
      - Active: True
      - Origin Network Instance: default
        Metric: 0
        Pref: 170
        Next-hop (Type): 'fe80::eaf3:75ff:fe05:52ad (direct)\nfe80::eaf3:75ff:fe05:52ae
(direct)'
        Next-hop Interface: ethernet-1/31.0 \nethernet-1/32.0
        Backup Next-hop (Type):
        Backup Next-hop Interface:
```

Example 47 **Route validation**

7.2 EDA validation

The validation steps shown below are Kubernetes CLI-based EDA validations; the UI workflow in the subsequent sections shows the UI validations.

Note: All the resources within EDA and Kubernetes exist within a specific namespace; as a result, while accessing the resources, either a -n <namespace name> or -A for all namespaces must be mentioned.

```
admin@server:~/nvd-hw/collapsed-spine$ kubectl get targetnodes -A
NAMESPACE               NAME            NODESECURITYPROFILE   STATUS   DHCP           ADDRESS
PORT
2-way-collapsed-spine   d2l-34-tor1     managed-tls           Ready    Acknowledged   192.168.80.103
57400
2-way-collapsed-spine   d2l-35-tor2     managed-tls           Ready    Acknowledged   192.168.80.104
57400
2-way-collapsed-spine   d3l-28-tor3     managed-tls           Ready    Acknowledged   192.168.80.105
57400
2-way-collapsed-spine   d3l-29-spine1   managed-tls           Ready    Acknowledged   192.168.80.101
57400
2-way-collapsed-spine   d3l-30-spine2   managed-tls           Ready    Acknowledged   192.168.80.102
57400
```

Example 48 **DHCP, gNMI port, and node status validation**

After the first phase is completed, EDA deploys an NPP pod per node that continues the onboarding process by performing a NOS status check and sync, then orchestrating the fabric by configuring the nodes. The expected states are that ONBOARDED is “true”, NPP is “Connected”, and NODE is “Synced” with the correct SR Linux version.

```
admin@server:~/nvd-hw/collapsed-spine$ kubectl get toponodes -A
NAMESPACE               NAME            PLATFORM       VERSION   OS    ONBOARDED   MODE     NPP
NODE     AGE
2-way-collapsed-spine   d2l-34-tor1     7220 IXR-D2L   25.3.2    srl   true        normal   Connected
Synced   8d
2-way-collapsed-spine   d2l-35-tor2     7220 IXR-D2L   25.3.2    srl   true        normal   Connected
Synced   8d
2-way-collapsed-spine   d3l-28-tor3     7220 IXR-D3L   25.3.2    srl   true        normal   Connected
Synced   8d
2-way-collapsed-spine   d3l-29-spine1   7220 IXR-D3L   25.3.2    srl   true        normal   Connected
Synced   8d
2-way-collapsed-spine   d3l-30-spine2   7220 IXR-D3L   25.3.2    srl   true        normal   Connected
Synced   8d
```

Example 49 **OS version, ZTP onboarding status validation**

Each of the resources cataloged via the **kubectl get** command can be viewed in further detail via the **kubectl describe** command; this command also provides verbose information about probable failure causes.

```
admin@server:~/nvd-hw/collapsed-spine$ kubectl describe targetnodes d3l-29-spine1 -n 2-
way-collapsed-spine
Name:         d3l-29-spine1
Namespace:    2-way-collapsed-spine
Labels:       eda.nokia.com/name=d3l-29-spine1
              eda.nokia.com/role=collapsed-spine
              eda.nokia.com/security-profile=managed
              eda.nokia.com/source=derived
Annotations:  <none>
API Version:  core.eda.nokia.com/v1
Kind:         TargetNode
Metadata:
  Creation Timestamp:  2025-06-26T10:03:38Z
  Generation:          1
  Managed Fields:
    API Version:  core.eda.nokia.com/v1
    Fields Type:  FieldsV1
    fieldsV1:
      f:metadata:
        f:labels:
          .:
          f:eda.nokia.com/name:
          f:eda.nokia.com/role:
          f:eda.nokia.com/security-profile:
          f:eda.nokia.com/source:
      f:spec:
        .:
        f:address:
        f:dhcp4:
          .:
          f:address:
          f:options:
```

```
        f:operatingSystem:
        f:platform:
        f:port:
        f:serialNumber:
        f:versionMatch:
        f:versionPath:
    Manager:      engine
    Operation:    Update
    Time:         2025-06-27T06:03:07Z
    API Version:  core.eda.nokia.com/v1
    Fields Type:  FieldsV1
    fieldsV1:
      f:status:
        .:
        f:bootstrapStatus:
        f:bootstrapStatusReason:
        f:dhcpStatus:
        f:tlsStatus:
          .:
          f:nodeSecurityProfile:
          f:tls:
            .:
            f:csrParams:
              .:
              f:certificateValidity:
              f:city:
              f:country:
              f:csrSuite:
              f:org:
              f:orgUnit:
              f:san:
                .:
                f:dns:
                f:ips:
            f:state:
          f:issuerRef:
    Manager:         manager
    Operation:       Update
    Subresource:     status
    Time:            2025-07-02T04:47:14Z
  Resource Version:  1647597
  UID:               e1b3bccf-6e3e-4de8-ba9d-b98dba89c46c
Spec:
  Address:  192.168.80.101
  dhcp4:
    Address:  192.168.80.101
    Options:
      Option:  3-Router
      Value:
        192.168.80.254
      Option:  51-IPAddressLeaseTime
      Value:
        604800
      Option:  1-SubnetMask
      Value:
        255.255.255.0
      Option:  67-BootfileName
      Value:
        http://192.168.80.7:80/core/httpproxy/v1/asvr/2-way-collapsed-spine/init-
base/bootscript-d3l-29-spine1/d3l-29-spine1-provision.py
  Operating System:  srl
  Platform:          7220 IXR-D3L
  Port:              57400
```

```
  Serial Number:     <omitted>
  Version Match:     v25\.3\.2.*
  Version Path:      .system.information.version
Status:
  Bootstrap Status:         Ready
  Bootstrap Status Reason:  onboard success
  Dhcp Status:              Acknowledged
  Tls Status:
    Node Security Profile:  managed-tls
    Tls:
      Csr Params:
        Certificate Validity:  2160h0m0s
        City:                  Sunnyvale
        Country:               US
        Csr Suite:
CSRSUITE_X509_KEY_TYPE_RSA_2048_SIGNATURE_ALGORITHM_SHA_2_256
        Org:                   NI
        Org Unit:              EDA
        San:
          Dns:
            d31-29-spine1
          Ips:
            192.168.80.101
        State:     California
      Issuer Ref:  eda-node-issuer
Events:            <none>
```

Example 50 **Target node description**

```
admin@server:~/nvd-hw/collapsed-spine$ kubectl describe toponodes d31-29-spine1 -n 2-way-
collapsed-spine
Name:         d31-29-spine1
Namespace:    2-way-collapsed-spine
Labels:       eda.nokia.com/name=d31-29-spine1
              eda.nokia.com/role=collapsed-spine
              eda.nokia.com/security-profile=managed
Annotations:  <none>
API Version:  core.eda.nokia.com/v1
Kind:         TopoNode
Metadata:
  Creation Timestamp:  2025-06-26T10:03:36Z
  Generation:          2
  Managed Fields:
    API Version:  core.eda.nokia.com/v1
    Fields Type:  FieldsV1
    fieldsV1:
      f:spec:
        f:onBoarded:
        f:productionAddress:
    Manager:      manager
    Operation:    Update
    Time:         2025-06-26T10:03:38Z
    API Version:  core.eda.nokia.com/v1
    Fields Type:  FieldsV1
    fieldsV1:
      f:metadata:
        f:annotations:
          .:
          f:kubectl.kubernetes.io/last-applied-configuration:
        f:labels:
          .:
          f:eda.nokia.com/name:
```

```
          f:eda.nokia.com/role:
          f:eda.nokia.com/security-profile:
      f:spec:
        .:
        f:nodeProfile:
        f:npp:
          .:
          f:mode:
        f:operatingSystem:
        f:platform:
        f:serialNumber:
        f:version:
    Manager:      kubectl-client-side-apply
    Operation:    Update
    Time:         2025-06-27T06:03:06Z
    API Version:  core.eda.nokia.com/v1
    Fields Type:  FieldsV1
    fieldsV1:
      f:status:
        .:
        f:node-details:
        f:node-state:
        f:npp-details:
        f:npp-pod:
        f:npp-state:
        f:operatingSystem:
        f:platform:
        f:version:
    Manager:         engine
    Operation:       Update
    Subresource:     status
    Time:            2025-07-02T04:46:53Z
  Resource Version:  1647500
  UID:               ed3119ab-fbd3-4ee7-97cd-4be987d3a742
Spec:
  Node Profile:  real-srlinux-25.3.2
  Npp:
    Mode:            normal
  On Boarded:        true
  Operating System:  srl
  Platform:          7220 IXR-D3L
  Production Address:
  Serial Number:  <omitted>
  Version:        25.3.2
Status:
  Node - Details:    192.168.80.101:57400
  Node - State:      Synced
  Npp - Details:     10.244.5.12:50057
  Npp - Pod:         eda-npp-0
  Npp - State:       Connected
  Operating System:  srl
  Platform:          7220 IXR-D3L
  Version:           25.3.2
Events:              <none>
```

Example 51 **TopoNode description**

EDA has a unique ability to determine the operational state of various components of the fabric on the CLI, from interfaces to VLANs to VRFs. The network administrator can get the overall state of the fabric via CLI and GUI. The following examples demonstrate these validations (the **edactl** commands shown are executed from within the *eda-toolbox* pod).

```
root in on eda-toolbox-696f47749d-z457h /eda
→  edactl get interfaces.interfaces.eda.nokia.com -A
NAMESPACE               NAME                      ENABLED   OPERATIONAL STATE   SPEED
LAST CHANGE
2-way-collapsed-spine   spine1-ethernet-1-25      true      up                  100G
2025-07-02T04:43:34.057Z
2-way-collapsed-spine   spine1-ethernet-1-26      true      up                  100G
2025-07-02T04:43:34.074Z
2-way-collapsed-spine   spine1-ethernet-1-29      true      up                  100G
2025-07-02T04:43:34.093Z
2-way-collapsed-spine   spine1-ethernet-1-31      true      up                  100G
2025-07-02T05:36:35.092Z
2-way-collapsed-spine   spine1-ethernet-1-32      true      up                  100G
2025-07-02T05:36:35.118Z
2-way-collapsed-spine   spine1-ixia-1             true      up                  100G
2025-07-02T04:41:55.802Z
2-way-collapsed-spine   spine1-ixia-2             true      up                  100G
2025-07-02T04:43:35.803Z
2-way-collapsed-spine   spine1-spine2-tor1-lag    true      up                  100G
2025-07-02T05:37:25.664Z
2-way-collapsed-spine   spine1-spine2-tor2-lag    true      up                  100G
2025-07-02T05:37:25.661Z
2-way-collapsed-spine   spine1-spine2-tor3-lag    true      up                  100G
2025-07-02T05:37:25.775Z
2-way-collapsed-spine   spine1-spine2-tor4-lag    true      degraded            100G
2025-07-02T05:37:27.899Z
2-way-collapsed-spine   spine2-ethernet-1-25      true      up                  100G
2025-07-02T05:37:23.646Z
2-way-collapsed-spine   spine2-ethernet-1-26      true      up                  100G
2025-07-02T05:37:23.646Z
2-way-collapsed-spine   spine2-ethernet-1-30      true      up                  100G
2025-07-02T05:37:23.663Z
2-way-collapsed-spine   spine2-ethernet-1-31      true      up                  100G
2025-07-02T05:35:43.807Z
2-way-collapsed-spine   spine2-ethernet-1-32      true      up                  100G
2025-07-02T05:35:43.823Z
2-way-collapsed-spine   spine2-ixia-1             true      up                  100G
2025-07-02T05:35:45.319Z
2-way-collapsed-spine   spine2-ixia-2             true      up                  100G
2025-07-02T05:37:25.892Z
2-way-collapsed-spine   tor1-ethernet-1-51        true      up                  100G
2025-07-02T04:54:44.072Z
2-way-collapsed-spine   tor1-ethernet-1-52        true      up                  100G
2025-07-02T05:49:24.934Z
2-way-collapsed-spine   tor1-ixia                 true      up                  100G
2025-07-02T04:09:39.642Z
2-way-collapsed-spine   tor1-spine-lag            true      up                  200G
2025-07-02T05:49:24.934Z
2-way-collapsed-spine   tor2-ethernet-1-51        true      up                  100G
2025-07-02T04:42:02.304Z
2-way-collapsed-spine   tor2-ethernet-1-52        true      up                  100G
2025-07-02T05:36:43.136Z
2-way-collapsed-spine   tor2-ixia                 true      up                  100G
2025-06-27T16:27:47.978Z
2-way-collapsed-spine   tor2-spine-lag            true      up                  200G
2025-07-02T05:36:43.136Z
2-way-collapsed-spine   tor3-ethernet-1-29        true      up                  100G
2025-07-02T04:46:37.527Z
2-way-collapsed-spine   tor3-ethernet-1-30        true      up                  100G
2025-07-02T05:41:18.387Z
```

```
2-way-collapsed-spine   tor3-ixia                true      up                  100G
2025-06-28T05:23:16.551Z
2-way-collapsed-spine   tor3-spine-lag           true      up                  200G
2025-07-02T05:41:18.387Z
```

Example 52 **Fabric-wide interface status**

```
root in on eda-toolbox-696f47749d-z457h /eda
➜  edactl get vlans.services.eda.nokia.com -A
NAMESPACE              NAME              BRIDGEDOMAIN   OPERDOWN SUBIF   TOTAL SUBIF
OPERATIONALSTATE   LASTCHANGE
2-way-collapsed-spine   tagged-v10         macvrf-v10     0                1            up
2025-07-02T05:41:39.000Z
2-way-collapsed-spine   tagged-v20         macvrf-v20     0                1            up
2025-07-02T04:46:58.000Z
2-way-collapsed-spine   tagged-v30         macvrf-v30     0                1            up
2025-07-02T05:41:39.000Z
2-way-collapsed-spine   tagged-v40         macvrf-v40     1                2            degraded
2025-07-02T05:42:54.000Z
2-way-collapsed-spine   tagged-v50         macvrf-v50     0                2            up
2025-07-02T05:42:52.000Z
2-way-collapsed-spine   tagged-v50-tor     v50-simple     0                2            up
2025-07-02T04:03:25.000Z
2-way-collapsed-spine   tagged-v70         macvrf-v70     0                2            up
2025-07-02T05:42:52.000Z
2-way-collapsed-spine   tagged-v70-tor     v70-simple     0                2            up
2025-07-01T06:31:43.000Z
2-way-collapsed-spine   untagged-v10       macvrf-v10     1                5            degraded
2025-07-02T05:42:54.000Z
2-way-collapsed-spine   untagged-v10-tor   v10-simple     0                2            up
2025-06-27T17:17:52.000Z
```

Example 53 **Fabric-wide VLAN status**

```
root in on eda-toolbox-696f47749d-z457h /eda
➜  edactl get bridgedomains.services.eda.nokia.com -A
NAMESPACE               NAME                   VNI     EVI   IMPORT TARGET   EXPORT TARGET
OPERDOWN SUBIF   TOTAL SUBIF   OPERATIONALSTATE   LASTCHANGE
2-way-collapsed-spine   macvrf-v10             10010   10    target:1:10     target:1:10
1                6             degraded           2025-07-02T05:42:54.000Z
2-way-collapsed-spine   macvrf-v20             10020   20    target:1:20     target:1:20
0                1             up                 2025-07-02T04:46:58.000Z
2-way-collapsed-spine   macvrf-v30             10030   30    target:1:30     target:1:30
0                1             up                 2025-07-02T05:41:39.000Z
2-way-collapsed-spine   macvrf-v40             10040   40    target:1:40     target:1:40
1                2             degraded           2025-07-02T05:42:54.000Z
2-way-collapsed-spine   macvrf-v50             10050   50    target:1:50     target:1:50
0                2             up                 2025-07-02T05:42:52.000Z
2-way-collapsed-spine   macvrf-v70             10070   70    target:1:70     target:1:70
0                2             up                 2025-07-02T05:42:52.000Z
2-way-collapsed-spine   untagged-v10-simple
0                0             down               2025-06-28T04:18:45.000Z
2-way-collapsed-spine   v10-simple
0                2             up                 2025-06-28T04:18:46.000Z
2-way-collapsed-spine   v50-simple
0                2             up                 2025-07-02T04:03:26.000Z
2-way-collapsed-spine   v70-simple
0                2             up                 2025-07-01T06:31:43.000Z
```

Example 54 **Fabric-wide bridge domain status**

```
root in on eda-toolbox-696f47749d-z457h /eda
→  edactl get irbinterfaces.services.eda.nokia.com -A
NAMESPACE               NAME      MTU    OPERATIONALSTATE   LASTCHANGE
2-way-collapsed-spine   irb-v10   9200   up                 2025-06-27T11:14:15.000Z
2-way-collapsed-spine   irb-v20   9200   up                 2025-07-02T04:46:58.000Z
2-way-collapsed-spine   irb-v30   9200   up                 2025-07-02T05:41:39.000Z
2-way-collapsed-spine   irb-v40   9200   up                 2025-06-28T04:54:14.000Z
2-way-collapsed-spine   irb-v50   9200   up                 2025-06-28T04:18:46.000Z
2-way-collapsed-spine   irb-v70   9200   up                 2025-06-28T05:23:28.000Z
```

Example 55 **Fabric-wide IRB status**

```
root in on eda-toolbox-696f47749d-z457h /eda
→  edactl get irbinterfaces.services.eda.nokia.com irb-v10 -n 2-way-collapsed-spine -o
json
{
    "kind": "IRBInterface",
    "metadata": {
        "name": "irb-v10",
        "namespace": "2-way-collapsed-spine"
    },
    "spec": {
        "arpTimeout": 250,
        "bridgeDomain": "macvrf-v10",
        "evpnRouteAdvertisementType": {
            "arpDynamic": true,
            "arpStatic": false,
            "ndDynamic": true,
            "ndStatic": false
        },
        "hostRoutePopulate": {
            "dynamic": false,
            "evpn": false,
            "static": false
        },
        "ipAddresses": [
            {
                "ipv4Address": {
                    "ipPrefix": "172.16.10.254/24",
                    "primary": true
                }
            },
            {
                "ipv6Address": {
                    "ipPrefix": "2001:db8:0:10::254/64",
                    "primary": true
                }
            }
        ],
        "ipMTU": 9200,
        "l3ProxyARPND": {
            "proxyARP": true,
            "proxyND": true
        },
        "learnUnsolicited": "BOTH",
        "router": "vrf1"
    },
    "status": {
        "interfaces": [
            {
                "enabled": true,
                "ipv4Addresses": [
```

```
                    {
                        "ipPrefix": "172.16.10.254/24",
                        "primary": true
                    }
                ],
                "ipv6Addresses": [
                    {
                        "ipPrefix": "2001:db8:0:10::254/64",
                        "primary": true
                    }
                ],
                "lastChange": "2025-07-02T05:35:43.579Z",
                "node": "d3l-30-spine2",
                "nodeInterface": "irb0.0",
                "operatingSystem": "srl",
                "operationalState": "up"
            },
            {
                "enabled": true,
                "ipv4Addresses": [
                    {
                        "ipPrefix": "172.16.10.254/24",
                        "primary": true
                    }
                ],
                "ipv6Addresses": [
                    {
                        "ipPrefix": "2001:db8:0:10::254/64",
                        "primary": true
                    }
                ],
                "lastChange": "2025-07-02T04:41:53.964Z",
                "node": "d3l-29-spine1",
                "nodeInterface": "irb0.0",
                "operatingSystem": "srl",
                "operationalState": "up"
            }
        ],
        "lastChange": "2025-06-27T11:14:15.000Z",
        "operationalState": "up"
    }
}
```


Example 56 **IRB description**

```
root in on eda-toolbox-696f47749d-z457h /eda
➜  edactl get routers.services.eda.nokia.com -A
NAMESPACE               NAME   VNI     EVI   IMPORT TARGET   EXPORT TARGET
OPERATIONALSTATE   LASTCHANGE
2-way-collapsed-spine   vrf1   10500   500   target:1:500    target:1:500    up
2025-07-02T05:41:39.000Z
```

Example 57 **VRF description and state**

You can use **kubectl get transactionresults -A** to view all EDA transactions from a Kubernetes view. You can view details of a specific transaction using **kubectl describe transactionresults [***transaction-name***] -n eda-system**.

```
admin@server:~/nvd-hw/collapsed-spine$ kubectl get transactionresults -A
NAMESPACE    NAME                      RESULT   AGE     DRYRUN   DESCRIPTION
eda-system   transaction-000000034   OK       8d               startup - loading embedded
manifests
eda-system   transaction-000000035   OK       8d               startup - initial load from
git
eda-system   transaction-000000036   OK       8d               startup - audit from
kubernetes

*snip*

eda-system   transaction-000000062   OK       8d
eda-system   transaction-000000063   OK       8d
eda-system   transaction-000000064   OK       8d
eda-system   transaction-000000065   OK       8d
eda-system   transaction-000000066   OK       8d
eda-system   transaction-000000067   OK       8d
eda-system   transaction-000000068   OK       8d
eda-system   transaction-000000069   OK       8d
eda-system   transaction-000000070   OK       8d
eda-system   transaction-000000071   OK       8d
eda-system   transaction-000000072   OK       8d
eda-system   transaction-000000073   OK       8d
eda-system   transaction-000000074   OK       8d
eda-system   transaction-000000075   OK       8d
eda-system   transaction-000000076   OK       8d
eda-system   transaction-000000077   OK       8d
eda-system   transaction-000000078   OK       8d
eda-system   transaction-000000079   OK       8d
eda-system   transaction-000000080   OK       8d
eda-system   transaction-000000081   OK       8d
eda-system   transaction-000000082   OK       8d
eda-system   transaction-000000083   OK       8d
eda-system   transaction-000000084   OK       8d
eda-system   transaction-000000085   OK       8d
eda-system   transaction-000000086   OK       8d
eda-system   transaction-000000087   OK       8d
eda-system   transaction-000000088   OK       8d
eda-system   transaction-000000089   OK       7d22h
eda-system   transaction-000000090   OK       7d22h
eda-system   transaction-000000091   OK       7d22h
eda-system   transaction-000000092   OK       7d22h
eda-system   transaction-000000093   OK       7d22h
eda-system   transaction-000000094   OK       7d22h
eda-system   transaction-000000095   OK       7d22h
eda-system   transaction-000000096   OK       7d22h
eda-system   transaction-000000097   OK       7d22h
eda-system   transaction-000000098   OK       7d22h
eda-system   transaction-000000099   OK       7d22h
eda-system   transaction-000000100   OK       7d22h
eda-system   transaction-000000101   OK       7d22h
eda-system   transaction-000000102   OK       7d22h

*snip*
```

Example 58 **Transaction details by using kubectl tool**

The tools and utilities shown Section 7.1 “Network validation” and 7.2 “EDA validation” are critical in both orchestrating and troubleshooting the fabric for before Day-0 and post Day-2 operations.

8 Automation and orchestration

8.1 Digital twin with Containerlab

Digital twins are an integral part of Day-0 through Day-2 operations, providing the operations and deployment teams with the opportunity to continuously validate the look and feel of any deployment. These virtual fabrics also grant the ability to learn and play with technologies and designs—in this case, a prescriptive collapsed spine EVPN VXLAN fabric that has been validated and tuned to provide maximum efficiency and redundancy.

A digital twin of this NVD can be deployed using Containerlab and containerized SR Linux. The repository for this specific deployment, which offers deployment options with and without EDA, can be found at: https://github.com/nokia/nokia-validated-designs/tree/main/validated-designs/collapsed-spine.

The digital twin deployment with EDA includes:

- all manifest files required for an EDA-orchestrated digital twin of the collapsed spine EVPN VXLAN NVD
- a bash script (*deploy-collapsed-spine-nvd.sh*) that deploys the end-to-end fabric
- a bash script (*destroy-collapsed-spine-nvd.sh*) that destroys all resources created by the deployment script

The deployment script assumes the Containerlab topology (*2-way-collapsed-spine.clab.yaml*) is already deployed and healthy. With the script, the fabric is up and running in roughly 2 to 3 minutes, with end-to-end server communication.

9 Reference designs

Validated designs (NVDs) and reference designs have some notable differences. NVDs are designs or end-to-end solutions that are officially supported, tested on hardware, and recommended by Nokia. They include lifecycle management systems, migration paths, official configuration recommendations, and solution support by Nokia support and services organizations. In comparison, reference designs are meant to be design guides that demonstrate alternate design and product capabilities of the Nokia portfolio, but do not undergo solution-wide hardware testing and are not subjected to end-to-end lifecycle management and migration path testing like the NVDs.

With that disclaimer in place, the reference designs are also tested designs by deploying them as virtual fabrics using Containerlab. These are ensured to be functional designs and have deployable GitHub repositories that can be downloaded and customized for further use, testing and exploration.

Reference designs for the collapsed spine architecture can be found at: https://github.com/nokia/nokia-validated-designs/tree/main/reference-designs/4-way-collapsed-spine.