Day 3 – Data Center Lab

Search or jump to... ctrl+k

Home › Dashboards › DC Fabric Telemetry

Add | Last 5 minutes | 5s

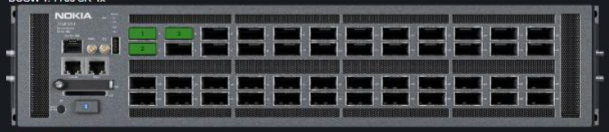
Fabric: leaf2:57400 | FabricServices: mgmt | DCI: dcgw2:57400 | DCIServices: 1

DC Fabric

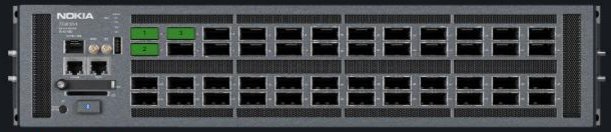
DC Fabric Throughput

Front panel view

DCGW-1: 7750 SR-1x



DCGW-2: 7750 SR-1x



Spine-1: 7220 IXR-D3L

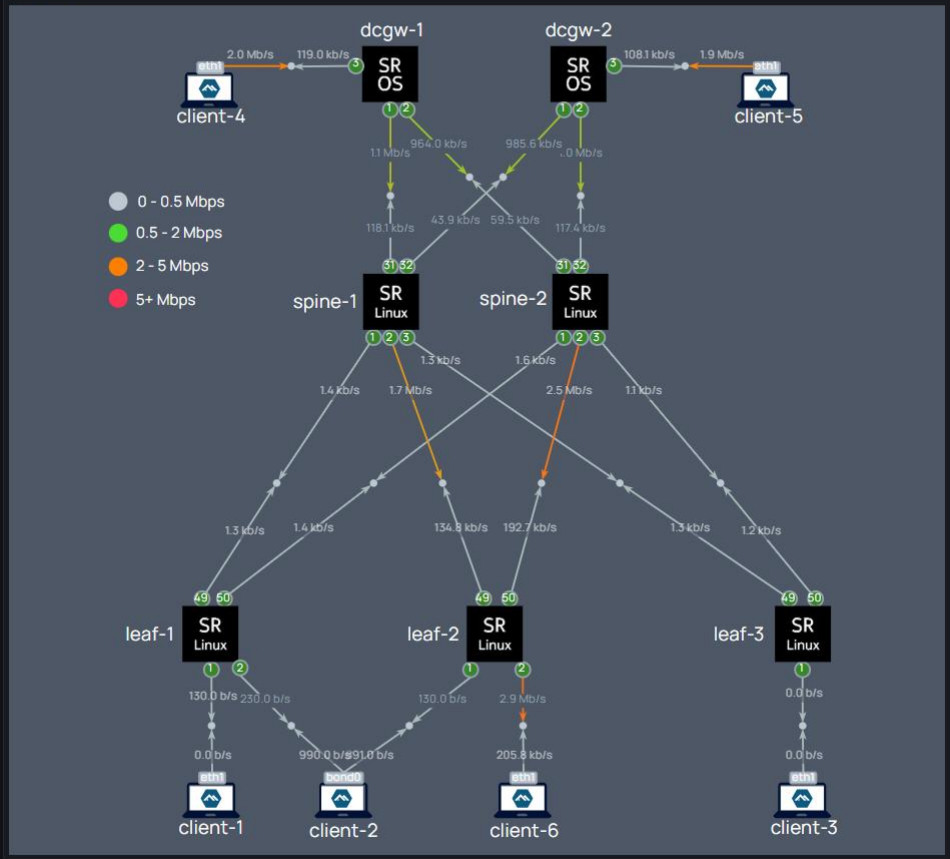
Spine-2: 7220 IXR-D3L

Leaf-1: 7220 IXR-D2L

Leaf-2: 7220 IXR-D2L

Leaf-3: 7220 IXR-D2L

Link Bandwidth



leaf2:57400

Deploying IXP DC lab

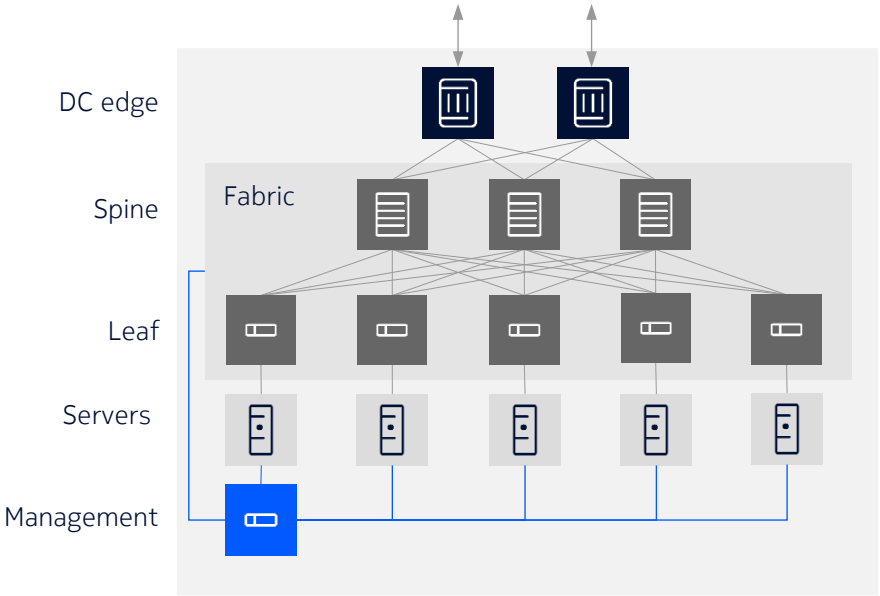
Agenda

- **Session #1**
  - Data Center topology
  - SR Linux introduction

- Session #2
  - EVPN

Data center network architectures

The industry has converged



Non-blocking fabrics

- IP and EVPN fabrics
- DC gateway or border leaf derivatives
- Collapsed core for edge DC
- Scale via super spines / pods

ASICs tailored per use case

- Range of different ASICs on the market
- Key properties: latency, programmability, port speed & density, feature set

OOB management

- Merchant silicon
- 1G/10G port speeds

Data center key foundations

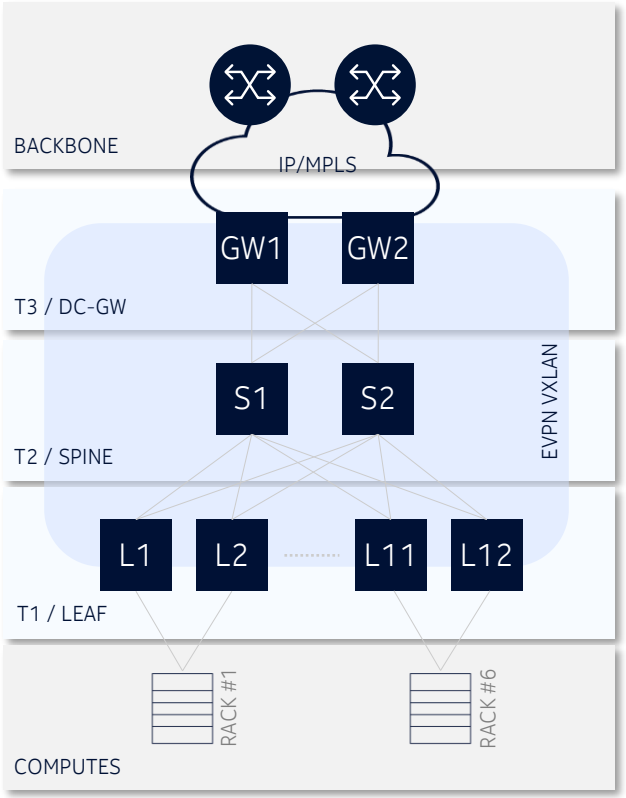
Goals and topology

Purpose of a data center

- Interconnect compute, storage and external networks at scale
- Ensure low-latency and high-throughput across the fabric
- Provide high availability through redundant design
- Enable secure communication between workloads

In practice, this is done through a Clos architecture.

Key factors like oversubscription ratio, fault tolerance, redundancy and scale will determine physical characteristics of the data center.



3-tier fabric with DC-GW

Data center key foundations

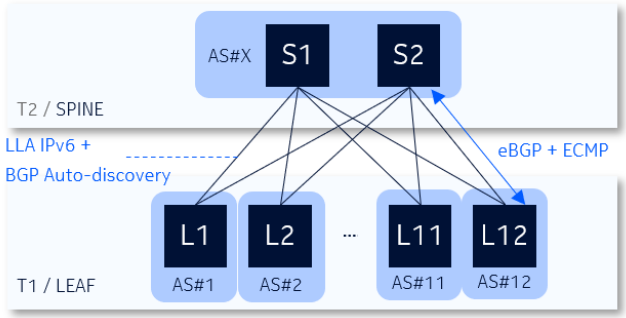
Underlay and control plane

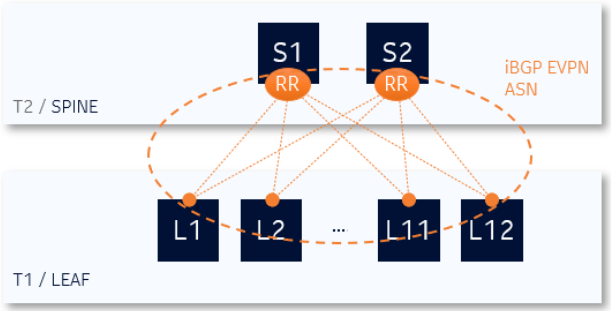
- Data centers have drastically evolved over the years and traffic is typically encapsulated over VxLAN using an EVPN control plane.

- In this workshop, we will focus on a modern data center architecture relying on the following protocols:

  - Underlay : IPv4 eBGP auto-discovery using link-local IPv6 addresses. Those BGP sessions will exchange VTEP addresses to create VxLAN tunnels between endpoints.

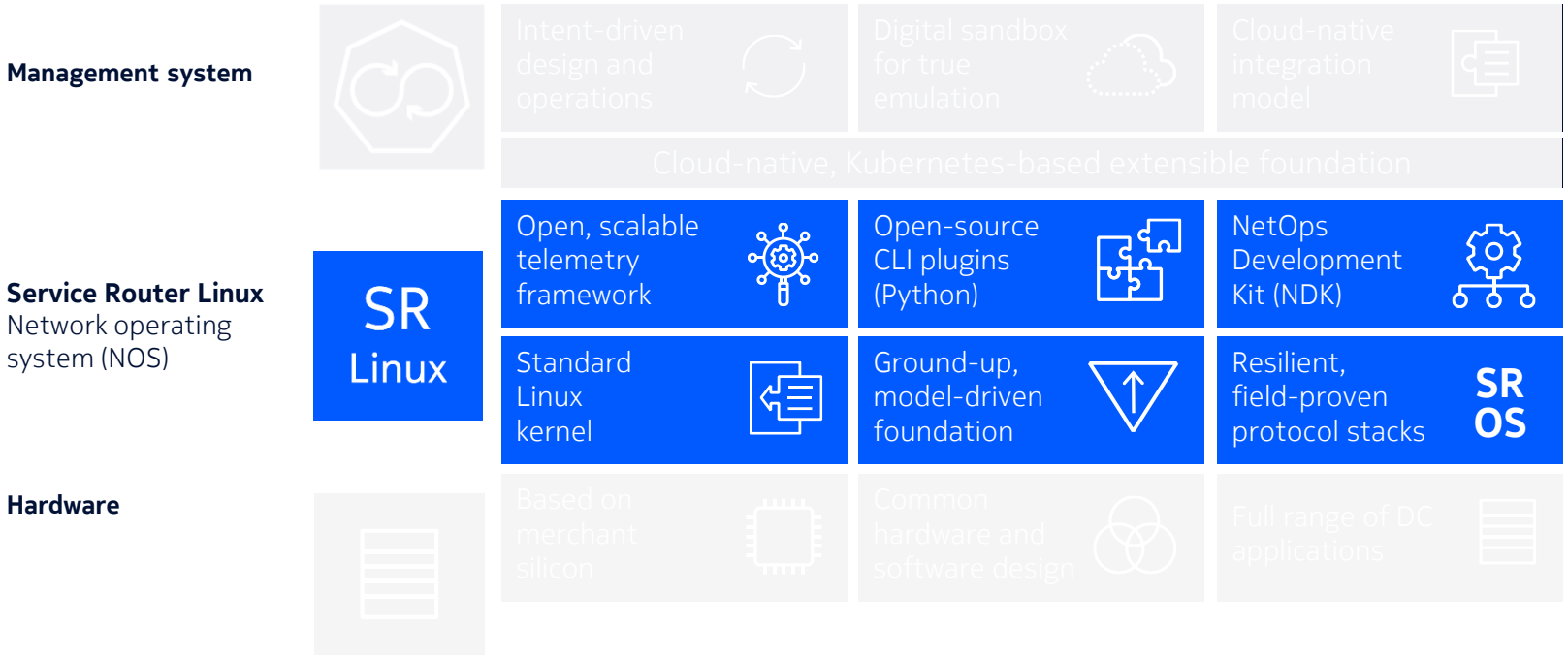
  - EVPN control plane : iBGP sessions exchange EVPN routes and rely on spines as route reflectors





Running data centres on Nokia SR Linux

A fresh look at the Operating System for Network devices



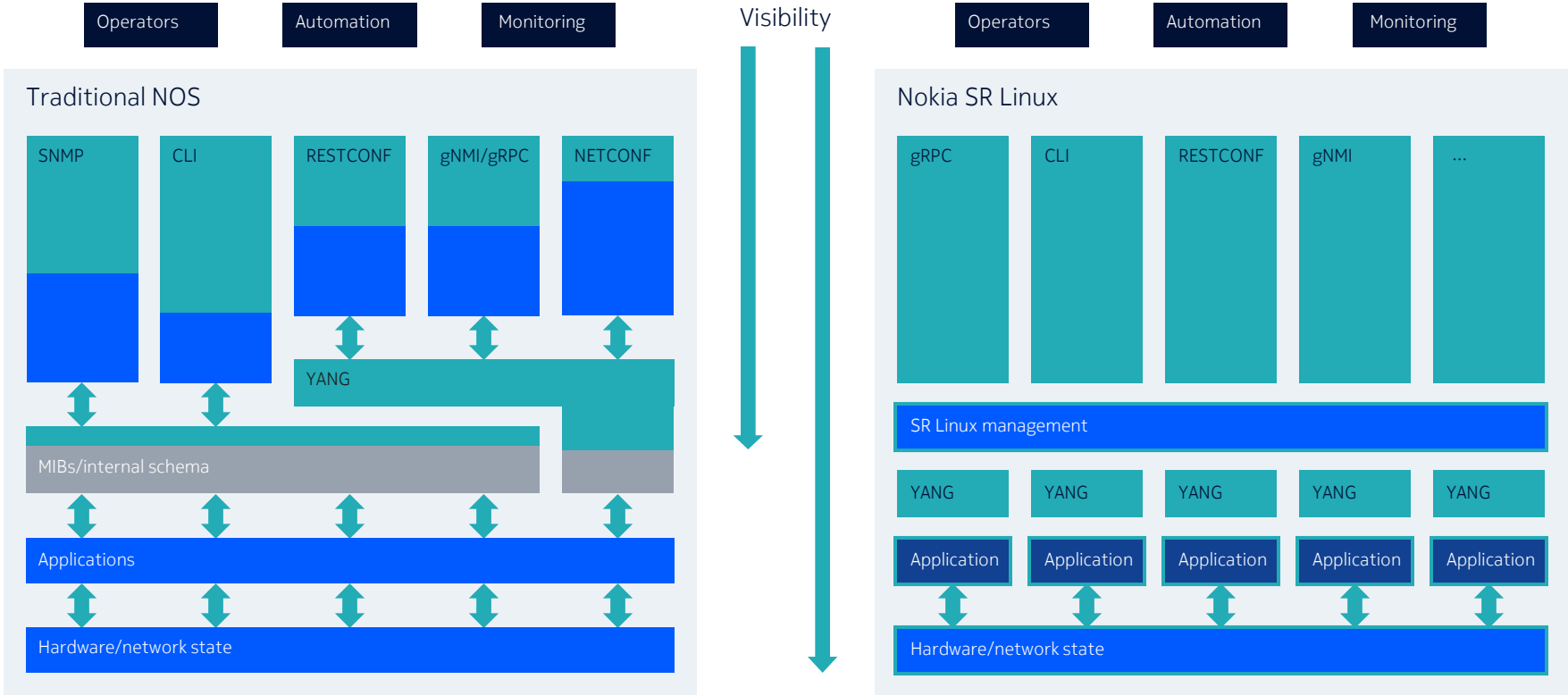
Resources and documentation

- Official documentation, publicly accessible
  No account needed, no license
  https://documentation.nokia.com/srlinux/

- “Get started” guide, tutorials, exercises for free
  https://learn.srlinux.dev/

- Discord – online community of SR Linux users
  https://discord.gg/tZvgjQ6PZf

Fully modelled Network OS

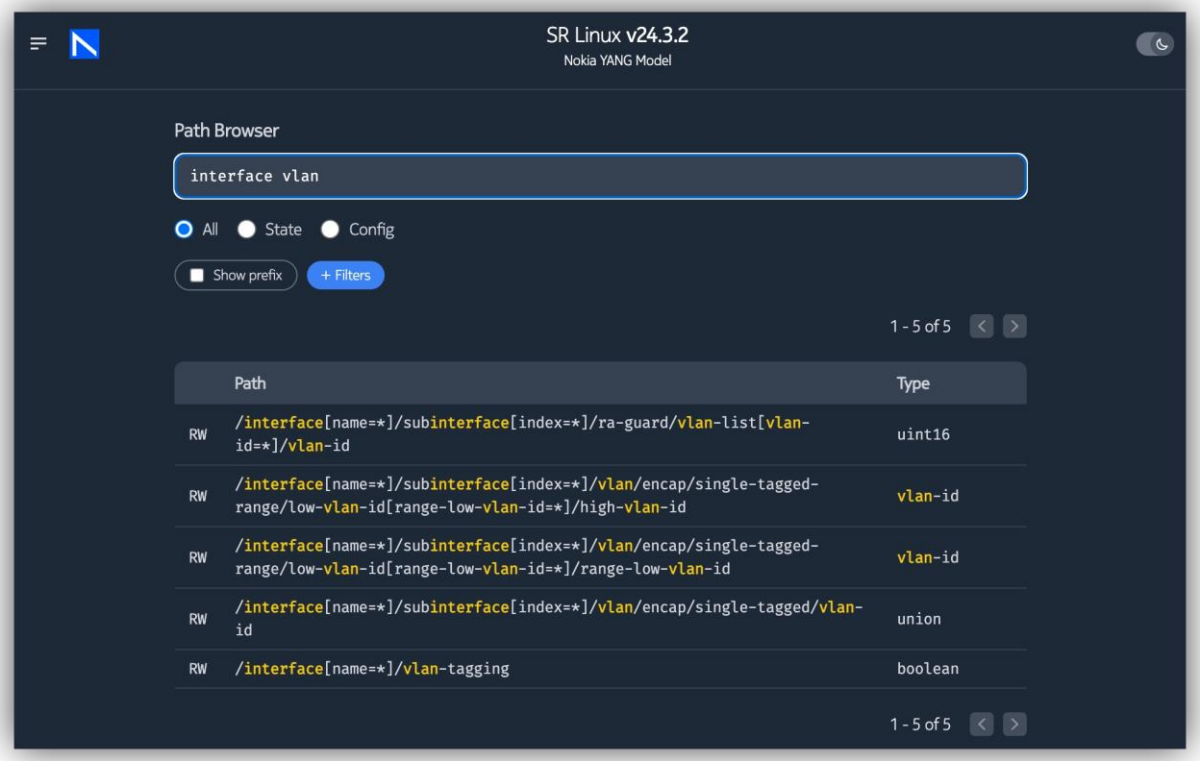


YANG Browser

Intuitive and fast path browsing

https://**yang.srlinux.dev**

- Responsive search experience for the YANG path
- Find leafs and substrees that match a search term
- Filtering through the YANG model



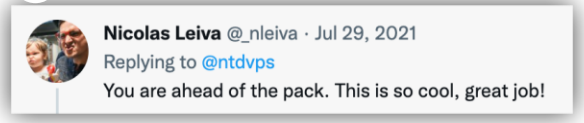
SR Linux Container Image

A key ingredient for our community

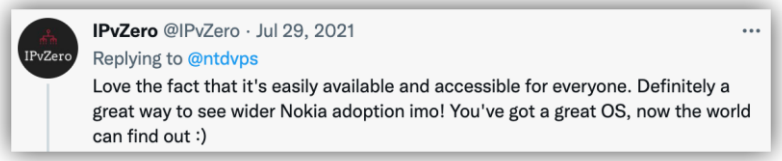
Open and free to use container image is a game changer for the industry.

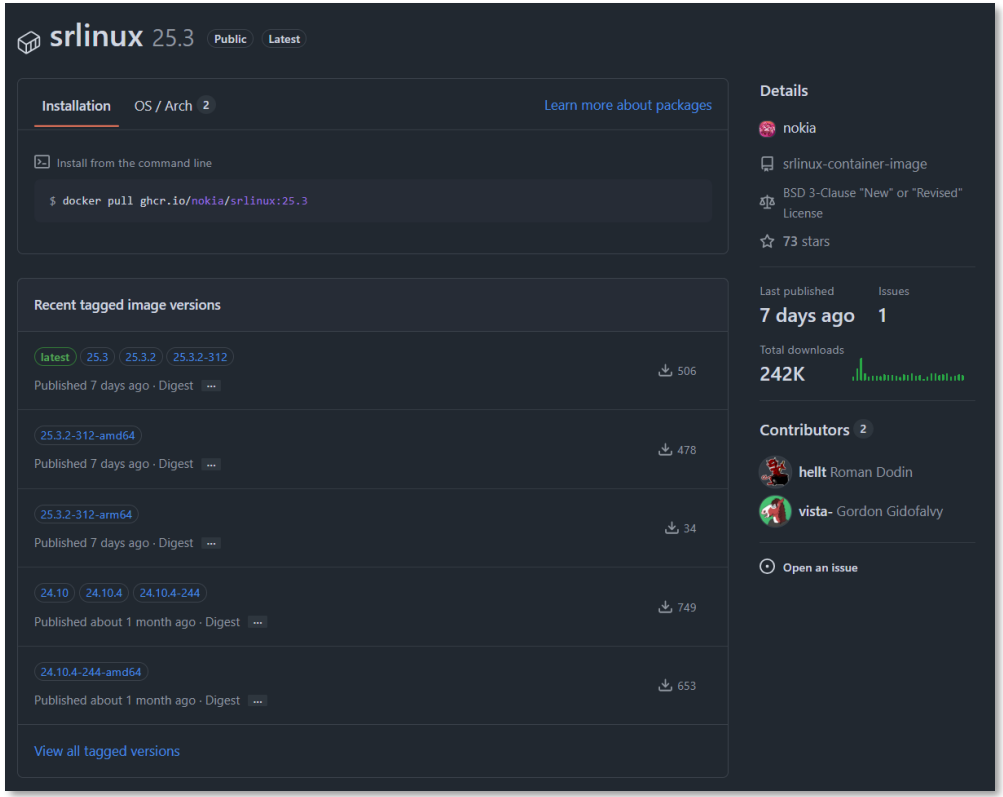
Effortless process to obtain the image made SR Linux so compelling to users.

redhat



CBI nuggets



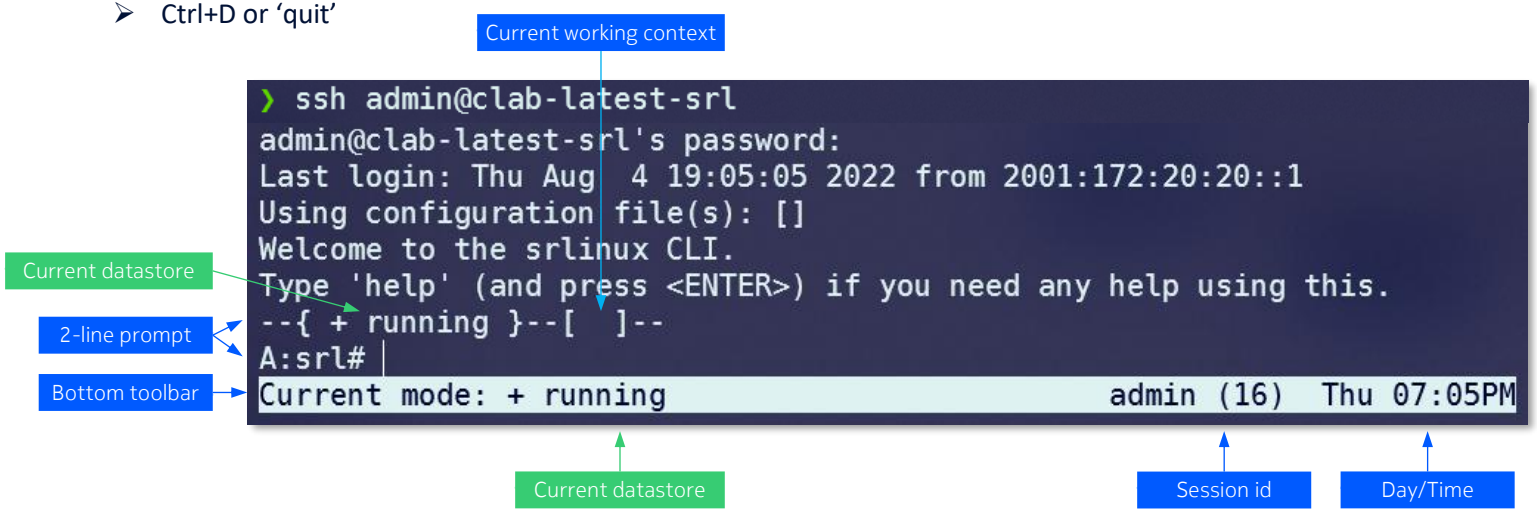



```
$ docker pull ghcr.io/nokia/srlinux:25.3.2
```


Diving into the CLI

Accessing the CLI

- Prompt and bottom toolbar can be customized
- Possibility to add pre- or post-login messages
- Exit CLI by typing
  - Ctrl+D or ‘quit’



Datastores

Overview

- Configuration and state information reside in datastores on the SR Linux device.
- The following 4 datastores are available based on RFC 8342[1]:
  - **Running** – contains the currently active configuration.
  - **Candidate** – contains a user-configurable version of the running datastore. Once committed, the candidate datastore becomes the running datastore.
  - **State** – contains the running configuration, plus dynamically added data such as operational state of interfaces or BGP peers added via auto-discovery, as well as session states and routing tables.
- **info** command is used to display information from a datastore.
  - **info from state** command (or entering the **info** command in state mode) displays configuration and statistics from the state datastore for the current context
  - **info from running** command (or the **info** command in running mode) displays configuration from the running datastore for the current context.

[1] Network management Datastore Architecture based on YANG

Diving into the CLI

Getting help from the CLI

- CLI provides context-based help:
  - Typing '?' shows all possible commands at that level
  - Typing '?' after a command displays the command usage

Current working context

Commands local to the current contest

```
--{ running }--[ system ]--
A:leaf1# ?
Local commands:
  aaa                 Top-level container for AAA services
  authentication      Container for protocol authentication options available system wide
  banner              Contains configuration and state related to system banners
  boot                Top-level container for configuration and state data related to booting the system
  bridge-table        system bridge-table information
  clock               Top-level container for system clock configuration and state
  configuration       Top-level container for configuration and state data related to the system
configuration
  dns                 Top-level container for DNS configuration and state
  ftp-server          Top-level container for FTP server configuration and state
  gnmi-server         Configures the gNMI server access API
  information         Top-level container for system information configuration and state
  json-rpc-server     Configures the JSON RPC access API
  lacp
<...>
Global commands:
  file                File related commands
  info                Show the values of all nodes and fields under the current context
  list                Show the keys of all nodes under the current context

Output modifier commands:
  #                   Comment out the rest of the line

--{ running }--[ system ]--
A:leaf1#

Current mode: running                                                    admin(20) Tue 11:18AM
```

Global commands

Help text extracted from YANG models description fields

```
--{ running }--[  ]--
A:leaf1# system ntp server ?
usage: server <address>

List of NTP servers to use for system clock synchronization

Positional arguments:
  address           [IP address, range IPv4|IPv6] IP address
of the NTP server, may be either IPv4 or IPv6

Current mode: running                                    admin(20) Tue 11:19AM
```

Diving into the CLI

CLI navigation

- Enter a <tab> to auto-complete the next command level

```
--{ running }--[  ]--
A:leaf1# system ssh-server
```

Non-ambiguous completion is displayed while typing

- If multiple options are available, a popup will appear

```
--{ running }--[  ]--
A:leaf1# interface ethernet-1/
                              ethernet-1/11
                              ethernet-1/55
                              ethernet-1/56
```

All possible options displayed in a popup box

- The options can be navigated

```
--{ running }--[  ]--
A:leaf1# system dns
                    aaa             boot            configuration   gnmi-server
                    authentication  bridge-table    dns             information
                    banner          clock           ftp-server      json-rpc-server
```

Once all options are displayed in the popup, use the arrow keys or tab to select one

Diving into the CLI

CLI navigation (2)

- To leave a context, use the **back** and **exit <all>** keywords
- **back** brings you back to the context you were before the last command
- **exit** leads you to the parent of the current context
- **exit all** leads you to the root context

```
--{ + running }--[ network-instance mgmt ]--
A:srl# protocols linux
--{ + running }--[ network-instance mgmt protocols linux ]--
A:srl# back
--{ + running }--[ network-instance mgmt ]--
A:srl# exit
--{ + running }--[  ]--
A:srl# network-instance mgmt protocols linux
--{ + running }--[ network-instance mgmt protocols linux ]--
A:srl# back
--{ + running }--[  ]--
A:srl# network-instance mgmt protocols linux
--{ + running }--[ network-instance mgmt protocols linux ]--
A:srl# exit all
--{ + running }--[  ]--
A:srl#
```

Diving into the CLI

Displaying information & possible formats

```
--{ + running }--[  ]--
A:srl# info | as <format>
             json
             table
             text
```

Text, by default

```
--{ + running }--[ network-instance mgmt ]--
A:srl# info
    type ip-vrf
    admin-state enable
    description "Management network instance"
    interface mgmt0.0 {
    }
    protocols {
        linux {
            import-routes true
            export-routes true
            export-neighbors true
        }
    }
```

JSON

```
--{ + running }--[ network-instance mgmt ]--
A:srl# info | as json
{
  "name": "mgmt",
  "type": "ip-vrf",
  "admin-state": "enable",
  "description": "Management network instance",
  "interface": [
    {
      "name": "mgmt0.0"
    }
  ],
  "protocols": {
    "linux": {
      "import-routes": true,
      "export-routes": true,
      "export-neighbors": true
    }
  }
}
```

Table

```
--{ + running }--[ network-instance mgmt ]--
A:srl# info | as table | filter protocols/linux fields *
+-------------+-------------+-------------+-------------+
|    Name     |  Protocols  |  Protocols  |  Protocols  |
|             |    linux    |    linux    |    linux    |
|             |   import-   |   export-   |   export-   |
|             |   routes    |   routes    |  neighbors  |
+=============+=============+=============+=============+
| mgmt        | true        | true        | true        |
+-------------+-------------+-------------+-------------+
```

Diving into the CLI

Displaying the default configuration

- Every context has a default configuration. When typing **info**, those default parameters are not displayed to facilitate reading.

- This implicit information can be displayed using **info detail**

```
--{ + running }--[ network-instance default protocols bgp ]--
A:srl# info
    autonomous-system 4200000005
    router-id 10.10.10.10
    group underlay-group {
    }
```

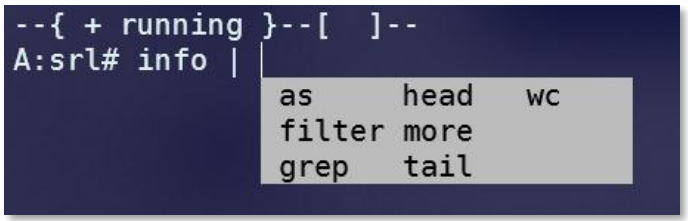
```
--{ + running }--[ network-instance default protocols bgp ]--
A:srl# info detail
    admin-state enable
    autonomous-system 4200000005
    local-preference 100
    router-id 10.10.10.10
    as-path-options {
        allow-own-as 0
        remove-private-as {
            mode disabled
            leading-only false
            ignore-peer-as false
        }
    }
    authentication {
    }
    convergence {
        min-wait-to-advertise 0
    }
    dynamic-neighbors {
        accept {
            max-sessions 0
        }
    }
    ebgp-default-policy {
        import-reject-all true
        export-reject-all true
    }
    failure-detection {
        enable-bfd false
        fast-failover true
    }
    graceful-restart {
        admin-state disable
        stale-routes-time 360
    }
```

```
    group underlay-group {
        admin-state enable
        next-hop-self false
        as-path-options {
        }
        authentication {
        }
        failure-detection {
        }
        multihop {
        }
        graceful-restart {
        }
        ipv4-unicast {
            prefix-limit {
                max-receive
                warning-thr
            }
        }
        ipv6-unicast {
            prefix-limit {
                max-receive
                warning-thr
            }
        }
        evpn {
            prefix-limit {
                max-receive
                warning-thr
            }
        }
        route-reflector {
        }
        send-community {
        }
        send-default-route {
            ipv4-unicast false
            ipv6-unicast false
        }
```

Diving into the CLI

CLI goodies

- Several tools are directly available on the CLI to manipulate the output
  - Among them, several well-known Linux tools : grep, head, tail, more, wc



Diving into the CLI

CLI goodies (2)

➢ Part of the tools taken from Linux : watch

```
--{ + running }--[  ]--
A:srl# watch show network-instance mgmt route-table ipv4-unicast summary
Every 2.0s: show network-instance mgmt route-table ipv4-unicast summary                                                     (Executions 14, Thu 07:57:50PM)

-----------------------------------------------------------------------------------------------------------------------------------------------------
IPv4 unicast route table of network instance mgmt
-----------------------------------------------------------------------------------------------------------------------------------------------------
+-----------------------+-------+------------+----------------------+----------------------+----------+---------+--------------+--------------+
|        Prefix         |  ID   | Route Type |     Route Owner      |        Active        |  Metric  |  Pref   |   Next-hop   |   Next-hop   |
|                       |       |            |                      |                      |          |         |    (Type)    |  Interface   |
+=======================+=======+============+======================+======================+==========+=========+==============+==============+
| 0.0.0.0/0             | 1     | dhcp       | dhcp_client_mgr      | True                 | 0        | 5       | 172.20.20.1  | mgmt0.0      |
|                       |       |            |                      |                      |          |         | (direct)     |              |
| 172.20.20.0/24        | 0     | linux      | linux_mgr            | False                | 0        | 5       | 172.20.20.0  | mgmt0.0      |
|                       |       |            |                      |                      |          |         | (direct)     |              |
| 172.20.20.0/24        | 1     | local      | net_inst_mgr         | True                 | 0        | 0       | 172.20.20.2  | mgmt0.0      |
|                       |       |            |                      |                      |          |         | (direct)     |              |
| 172.20.20.2/32        | 1     | host       | net_inst_mgr         | True                 | 0        | 0       | None         | None         |
|                       |       |            |                      |                      |          |         | (extract)    |              |
| 172.20.20.255/32      | 1     | host       | net_inst_mgr         | True                 | 0        | 0       | None         |              |
|                       |       |            |                      |                      |          |         | (broadcast)  |              |
+-----------------------+-------+------------+----------------------+----------------------+----------+---------+--------------+--------------+
-----------------------------------------------------------------------------------------------------------------------------------------------------
IPv4 routes total                    : 5
IPv4 prefixes with active routes     : 4
IPv4 prefixes with active ECMP routes: 0
-----------------------------------------------------------------------------------------------------------------------------------------------------
```


Diving into the CLI

CLI goodies (3)

- Monitoring specific YANG nodes is also possible with : monitor

```
--{ + state }--[ system aaa authentication ]--
A:srl# /monitor system gnmi-server
[2022-08-04 20:12:39.613263]: update /system/gnmi-server/admin-state:enable
[2022-08-04 20:12:39.613629]: update /system/gnmi-server/timeout:7200
[2022-08-04 20:12:39.613907]: update /system/gnmi-server/rate-limit:60
[2022-08-04 20:12:39.614055]: update /system/gnmi-server/session-limit:20
[2022-08-04 20:12:39.614166]: update /system/gnmi-server/commit-confirmed-timeout:0
[2022-08-04 20:12:39.614291]: update /system/gnmi-server/commit-save:false
[2022-08-04 20:12:39.614392]: update /system/gnmi-server/include-defaults-in-config-only-responses:false
[2022-08-04 20:12:39.614491]: update /system/gnmi-server/unix-socket/admin-state:disable
[2022-08-04 20:12:39.614591]: update /system/gnmi-server/unix-socket/oper-state:down
[2022-08-04 20:12:39.614691]: update /system/gnmi-server/unix-socket/use-authentication:true
[2022-08-04 20:12:39.614790]: update /system/gnmi-server/unix-socket/socket-path:
[2022-08-04 20:12:54.608966]: update /system/gnmi-server/admin-state:disable
[2022-08-04 20:13:04.961993]: update /system/gnmi-server/admin-state:enable
```

Diving into the CLI

Configuring and committing changes to SR Linux

- To modify the existing configuration, enter the candidate datastore, modify the configuration, and commit the changes.

```
--{ running }--[  ]--
A:admin@srl# enter candidate


--{ candidate shared default }--[  ]--
A:admin@srl#
```

After verifying, commit the changes to the running datastore

Apply your changes, and optionally check the differences with the **diff** command

```
--{ candidate shared default }--[  ]--
A:admin@srl# network-instance default


--{ * candidate shared default }--[ network-instance default ]--
A:admin@srl# description "Default instance for my INNOG8 Datacenter switch!"


--{ * candidate shared default }--[ network-instance default ]--
A:admin@srl# info
    description "Default instance for my INNOG8 Datacenter switch!"
```

```
--{ * candidate shared default }--[ network-instance default ]--
A:admin@srl# commit stay
All changes have been committed. Starting new transaction.


--{ + candidate shared default }--[ network-instance default ]--
A:admin@srl#
```

* : unsaved config in the candidate datastore

\+ : config in the running datastore not saved in the startup file

Diving into the CLI

Configuring and committing changes to SR Linux (2)

- Advanced commands can be used to configure or commit the configuration
  - Configuration can be loaded from existing startup, rescue, factory configurations, from checkpoints or files, or can be typed in json format directly.

```
--{ + candidate shared default }--[ network-instance default ]--
A:srl# load
            checkpoint json
            factory    rescue
            file       startup
```

  - When committing, multiple options are available :

```
--{ + candidate shared default }--[ network-instance default ]--
A:srl# commit
usage: commit

Apply all changes. Will update the applications if successful

Local commands:
  checkpoint        Save the configuration to a checkpoint after successful commit
  confirmed         Start confirmation timer (will revert changes if not confirmed)
  now               Leave the current context
  save              Save the configuration as startup configuration after successful commit and leave the current context
  stay              Stay in the current context and open new configuration session
  validate          Validate all changes
```

Diving into the CLI

Show command example

```
A:linx-spine-1# /show interface ethernet-1/7 detail
===============================================================================
Interface: ethernet-1/7
-------------------------------------------------------------------------------
  Description      : <None>
  Oper state       : up
  Down reason      : N/A
  Last change      : 1d18h45m53s ago, 3 flaps since last clear
  Speed            : 100G
  Flow control     : Rx is enabled
  MTU              : 8950
  VLAN tagging     : false
  VLAN TPID        : TPID_0X8100
  Queues           : 8 output queues supported, 6 used since the last clear
  Last stats clear: never
  Breakout mode    : false
-------------------------------------------------------------------------------
L2CP transparency rule for ethernet-1/7
-------------------------------------------------------------------------------
  Lldp              : trap-to-cpu-untagged
  Lacp              : trap-to-cpu-untagged
  xStp              : drop-tagged-and-untagged
  Dot1x             : drop-tagged-and-untagged
  Ptp               : drop-tagged-and-untagged
  Non-specified l2cp: false
-------------------------------------------------------------------------------
Traffic statistics for ethernet-1/7
-------------------------------------------------------------------------------
       counter             Rx              Tx
  Octets              21263736028626  21382073138556
  Unicast packets     15147017826     22762664074
  Broadcast packets   24              22
  Multicast packets   91312           91304
  Errored packets     0               0
  FCS error packets   0               N/A
  MAC pause frames    0               0
  Oversize frames     0               N/A
  Jabber frames       0               N/A
  Fragment frames     0               N/A
  CRC errors          0               N/A
-------------------------------------------------------------------------------
Traffic rate statistics for ethernet-1/7
-------------------------------------------------------------------------------
    units     Rx   Tx
  kbps rate   3    5
-------------------------------------------------------------------------------
Frame length statistics for ethernet-1/7
-------------------------------------------------------------------------------
  Frame length(Octets)       Rx            Tx
  64 bytes                76            72
  65-127 bytes            1778378       3696884
  128-255 bytes           1567842241    3933508496
  256-511 bytes           89            7070740861
  512-1023 bytes          18            78
  1024-1518 bytes         7             13227573
  1519+ bytes             13577488353   11741581436
-------------------------------------------------------------------------------
Transceiver detail for ethernet-1/7
-------------------------------------------------------------------------------
  Status           : Transceiver is present and operational
  Form factor      : QSFP28
  Channels used    : 4
  Connector type   : LC
  Vendor           : NOKIA
  Vendor part      : 3HE10550AARA01
  PMD type         : 100GBASE-LR4
  Fault condition : false
  Temperature      : 34
  Voltage          : 3.2322
===============================================================================
  Subinterface: ethernet-1/7.0
-------------------------------------------------------------------------------
  Description           : <None>
  Network-instance      : default
  Type                  : routed
  Oper state            : up
  Down reason           : N/A
  Last change           : 1d18h45m53s ago
  Encapsulation         : null
  IP MTU                : 8830
  Last stats clear      : never
  MAC duplication action: -
  IPv4 addr    : 100.70.0.143/31 (static, preferred, primary)
-------------------------------------------------------------------------------
```

Diving into the CLI

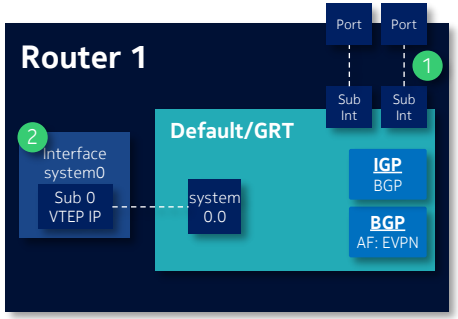
Show command example

```
---------------------------------------------------
ARP/ND summary for ethernet-1/7.0
---------------------------------------------------
  IPv4 ARP entries : 0 static, 1 dynamic
---------------------------------------------------
ACL filters applied to ethernet-1/7.0
---------------------------------------------------
     Summary       In    Out
  IPv4 ACL Name   none   none
  IPv6 ACL Name   none   none
---------------------------------------------------
QOS Policies applied to ethernet-1/7.0
---------------------------------------------------
      Summary         In        Out
  DSCP classifier   default   -
  DSCP rewrite      -         default
---------------------------------------------------
Traffic statistics for ethernet-1/7.0
---------------------------------------------------
     Statistics             Rx              Tx
  Packets             15147109154      22762487268
  Octets              21263713694717   21382050772480
  Discarded packets   268195           0
  Forwarded packets   15146840959      22762487268
  Forwarded octets    21263713694717   21382050772480
  CPM packets         -                0
  CPM octets          -                0
  Statistics   Rx   Tx
  Statistics   Rx   Tx
  Statistics   Rx   Tx
```


NOKIA

Configuring an interface on SR Linux

Basic IPv4 interface configuration



1

```
# info interface ethernet-1/55
    interface ethernet-1/55 {
        admin-state enable
        vlan-tagging true
        subinterface 1 {
            ipv4 {
                address 101.1.1.0/31 {
                }
            }
            ipv6 {
                address 2002::101:1:1:0/127 {
                }
            }
            vlan {
                encap {
                    single-tagged {
                        vlan-id 1
                    }
```

ethernet-1/55 is an uplink interface, i.e. towards the fabric

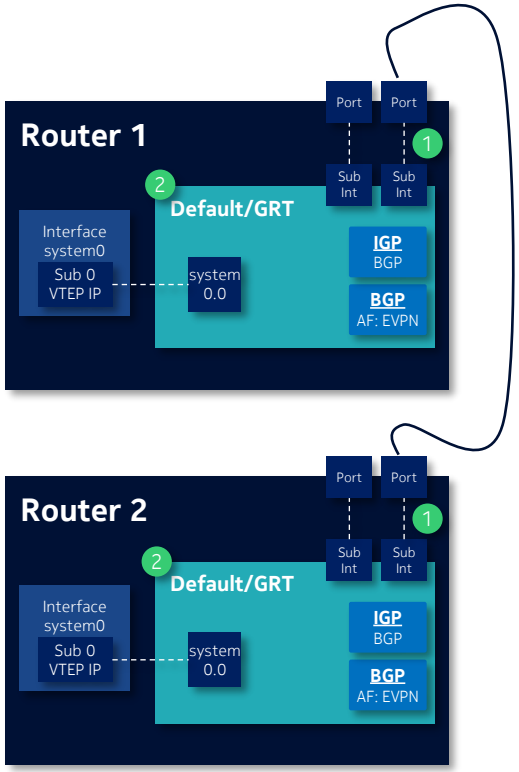
system0.0 is the loopback interface used to originate and terminate VxLAN packets

2

```
# info interface system0
    interface system0 {
        admin-state enable
        subinterface 0 {
            admin-state enable
            ipv4 {
                address 192.1.1.1/32 {
                }
            }
            ipv6 {
                address 2000::192:1:1:1/128 {
                }
            }
        }
    }
```

Configuring an interface on SR Linux

Link-local IPv6 address and BGP auto-discovery configuration



1

```
# info interface ethernet-1/49
 admin-state enable
   subinterface 0 {
      ipv6 {
         admin-state enable
         router-advertisement {
            router-role {
               admin-state enable
            }
         }
      }
   }
```

Create a subinterface and enable **router-advertisement**

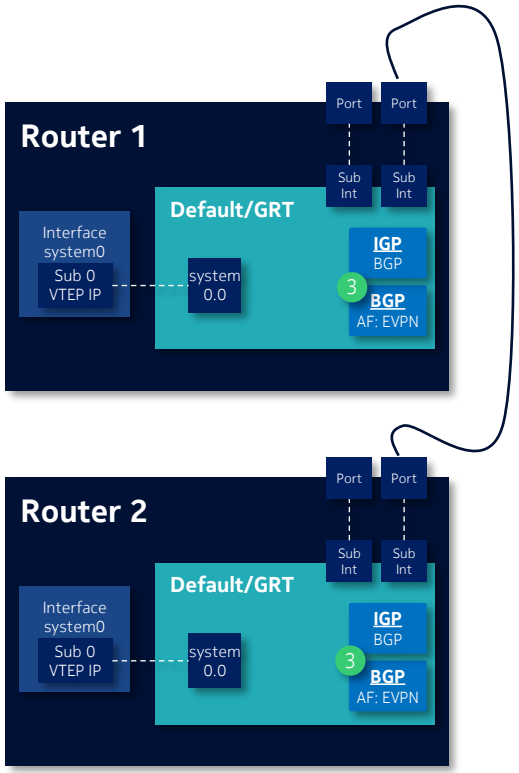
Add a subinterface to dynamic-neighbors in the BGP context to enable BGP unnumbered peers.

2

```
# info network-instance default
  admin-state enable
  router-id 192.1.1.1
   interface ethernet-1/49.0 {
   }
   interface system0.0 {
   }
   protocols {
      bgp {
         admin-state enable
         autonomous-system 65413
         router-id 10.0.1.3
         dynamic-neighbors {
            interface ethernet-1/49.0 {
               peer-group underlay
               allowed-peer-as [
                  65500
               ]
            }
```

Configuring an interface on SR Linux

Configuring BGP for the overlay using Route Reflection



3

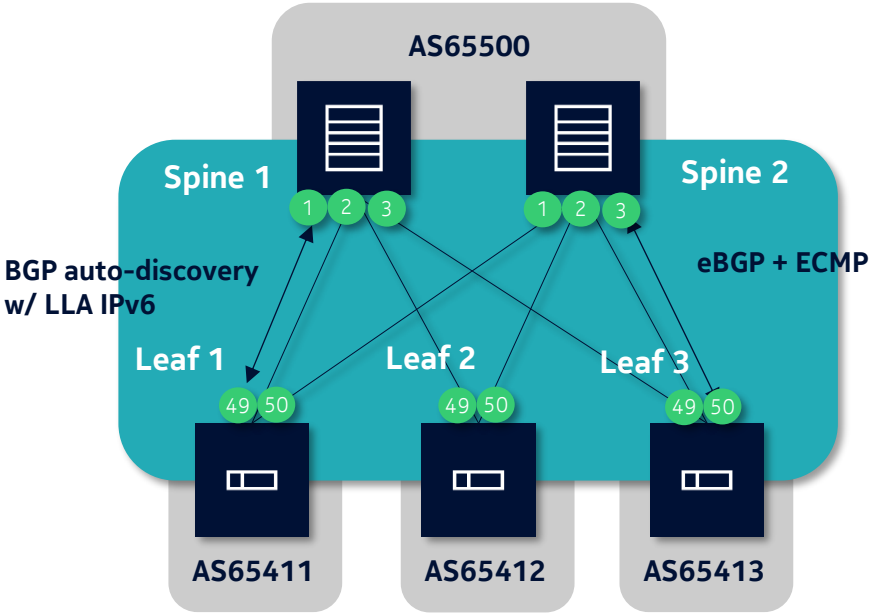
```
network-instance default {
    protocols {
        bgp {
            neighbor NEIGHBOR_IP {
                admin-state enable
                peer-group overlay
            }
            group overlay {
** configuration snip**
                route-reflector {
                    client true
                    cluster-id SYSTEM_IP
                }
            }
        }
    }
}
```

(i) Each iBGP session is explicitly defined, on both nodes.

(ii) Route reflection is typically enabled between leafs and spines, with spines acting as reflectors. Only spines need additional configuration to enable RR.
No specific configuration is required on leafs to be RR client.

Hands-on activity #1

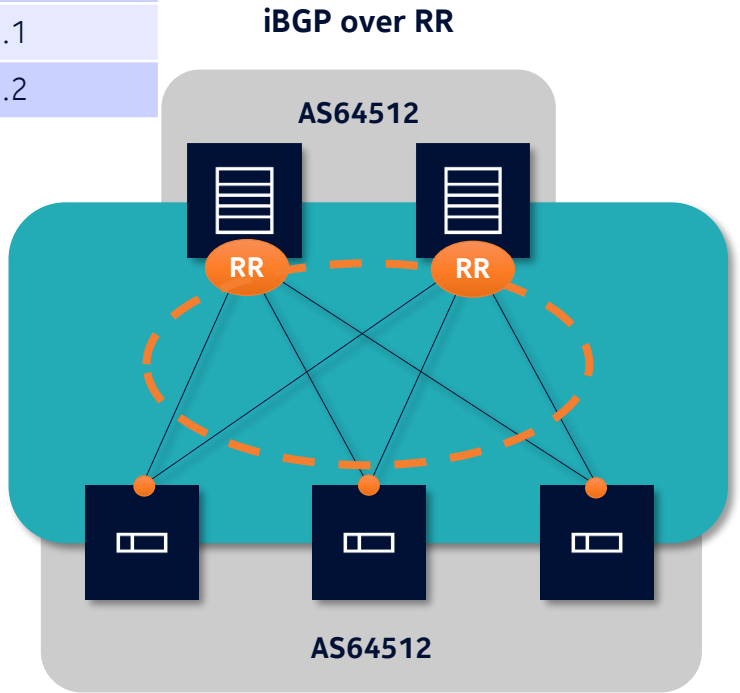
- This first activity will require you to set up the foundations of your containerised data center!
- **Task 1 – Underlay**
  - Our goal is to establish eBGP sessions using only link-local IPv6 addresses on the links between leafs and spines.
  - Based on the previous slides,
    - (1) make sure that router advertisements for IPv6 are enabled on the local interfaces
    - (2) configure BGP with dynamic neighbors. The configuration for the BGP group *underlay is* already available on the routers, make sure to use it!
  - Note: subinterfaces must be created and added to the default network-instance.
  - **Verify** that BGP sessions have been established by typing:
    - show network-instance default protocols bgp neighbor



Hands-on activity #1

Node	System IP
Leaf1	10.0.1.1
Leaf2	10.0.1.2
Leaf3	10.0.1.3
Spine1	10.0.2.1
Spine2	10.0.2.2

- **Task 2 – EVPN Control Plane**
  - Once your eBGP sessions have been established, the next step is to connect VTEPs together.
  - Each leaf and spine now need to be configured to establish iBGP sessions.
  - Spines will be used as route reflectors such that each leaf will only have two iBGP sessions in total, one towards each spine.
  - The configuration for BGP overlay group has been created but route reflection still needs to be enabled on the spine.
  - Verify that BGP sessions have been established by typing:
    - show network-instance default protocols bgp neighbor



Deploying IXP DC lab

Agenda

- Session #1
  - Data Center topology
  - SR Linux introduction

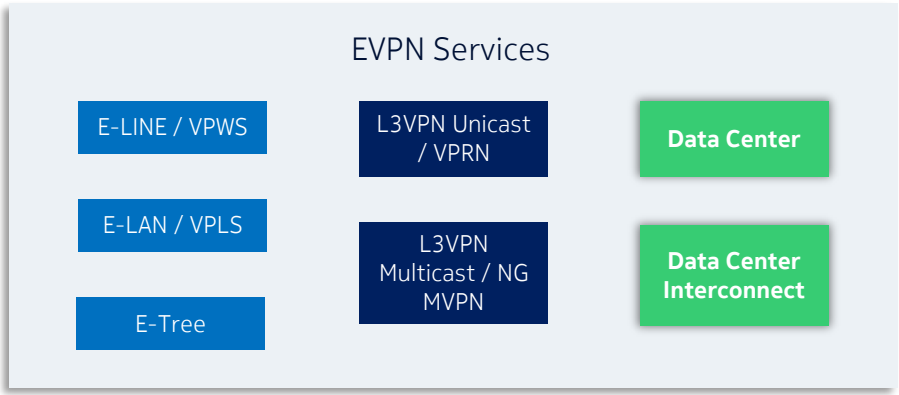
- **Session #2**
  - EVPN

A service model to rule them all

What is Ethernet VPN ?

- **VPN services** were traditionally delivered using **a different technology per service type** : for instance, BGP/LDP to deliver VPLS and VPWS, MP-BGP/MPLS to deliver IP VPNs, and BGP/PIM to deliver multicast VPNs. Combined together, these technologies **add complexity and increase the operational costs** for service providers.

- Ethernet virtual private network (**EVPN**) introduces a **unified model** for VPNs and cloud-based services, by providing a **control plane framework** that can deliver **any type of VPN services**.



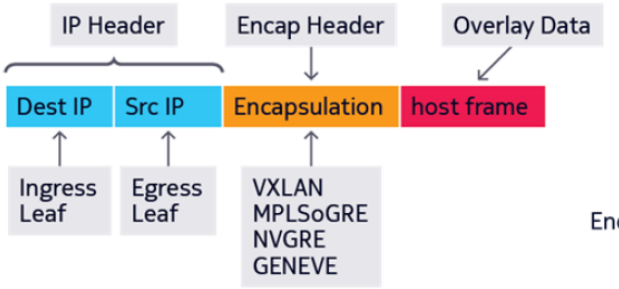
- Specifications for **overlays in data centers** are defined based on RFCs. Notably, they describe how to use EVPN across VXLAN or MPLS tunnels.

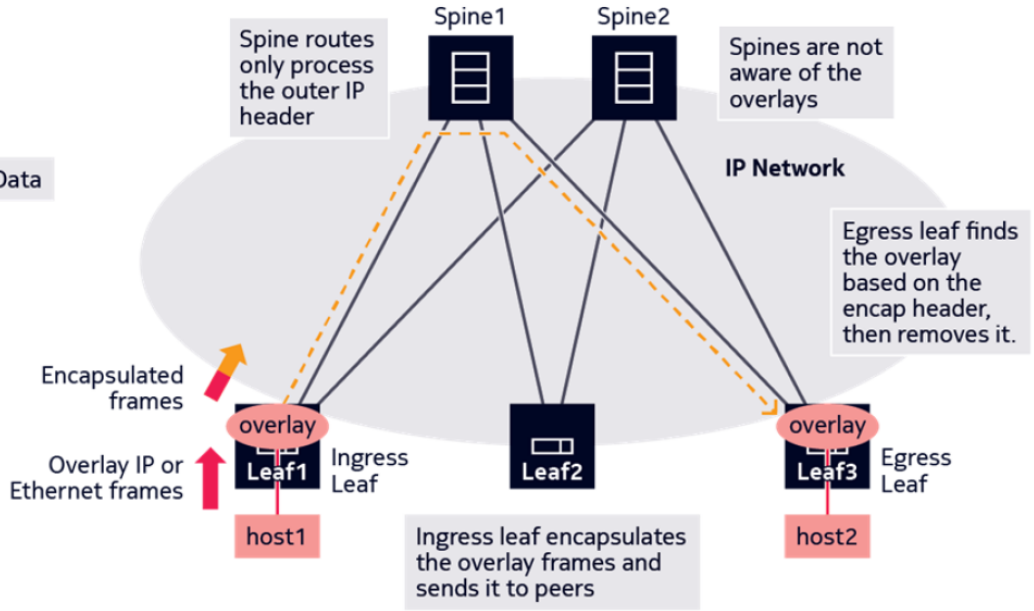
For further reference,
RFC 7432 - BGP MPLS-Based Ethernet VPN (ietf.org)
RFC 8365 - A Network Virtualization Overlay Solution Using Ethernet VPN (EVPN) (ietf.org)
RFC 9014 - Interconnect Solution for Ethernet VPN (EVPN) Overlay Networks (ietf.org)

EVPN – Supported Data Planes

EVPN supports multiple data plane technologies for tunneling overlay networks:

- Service Providers prefer MPLS (and PBB for very large-scale networks)
- IP-based encapsulation in the data center:
  - VXLAN is the most popular,
  - Also MPLSoGRE, NVGRE, GENEVE





EVPN – VXLAN in Data Centers

VXLAN – Virtual eXtensible Local Area Network:

- VXLAN is a Layer 2 overlay using an existing Layer 3 network infrastructure (underlay)
- VXLAN is defined in RFC 7348
- Was in use well before EVPN

VXLAN became a de-facto standard in data center networking:

- Allows the creation of L2 overlay networks that span the whole data center:
  - Scale up to 16M tenants (vs 4K with VLANs), isolating each overlay from each other
  - Seamless VM mobility within the data center
- Leverages the underlay IP networks:
  - To avoid loops (no more Spanning Tree Protocol)
  - To provide efficient multi-path load-balancing with ECMP
- However, RFC 7348 does not specify a control plane:
  - VXLAN uses Flood-and-Learn in the data plane
  - Requires static configuration to learn about other VXLAN endpoints

EVPN – VXLAN Terminology

VXLAN tunnels:

- To implement an overlay network, traffic is encapsulated with an extra header identifying the overlay. VXLAN is one of such encapsulation techniques.
- Encapsulated traffic flows between two end-points over the underlay network – hence the name 'tunnel'.

VXLAN Tunnel End Point (VTEP):

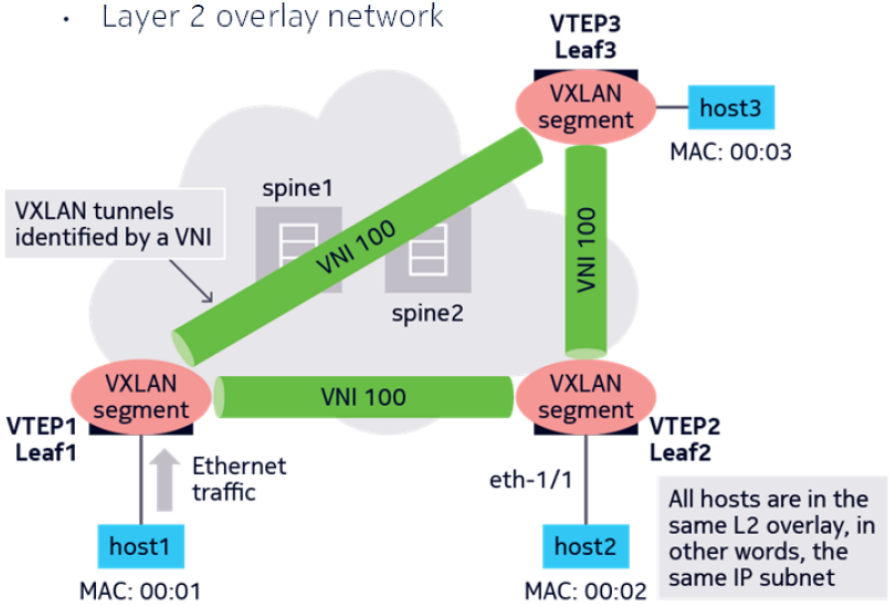
- Egress and Ingress point of the encapsulated traffic.
- Typically, the leaf routers
- The spine routers are not aware of the VXLAN tunnels

VXLAN Network Identifier (VNI):

- 24-bit integer uniquely identifying a VXLAN segment network-wide.

VXLAN segment:

- Layer 2 overlay network



EVPN – VXLAN Encapsulated Frame

VXLAN encapsulates Ethernet in IP:

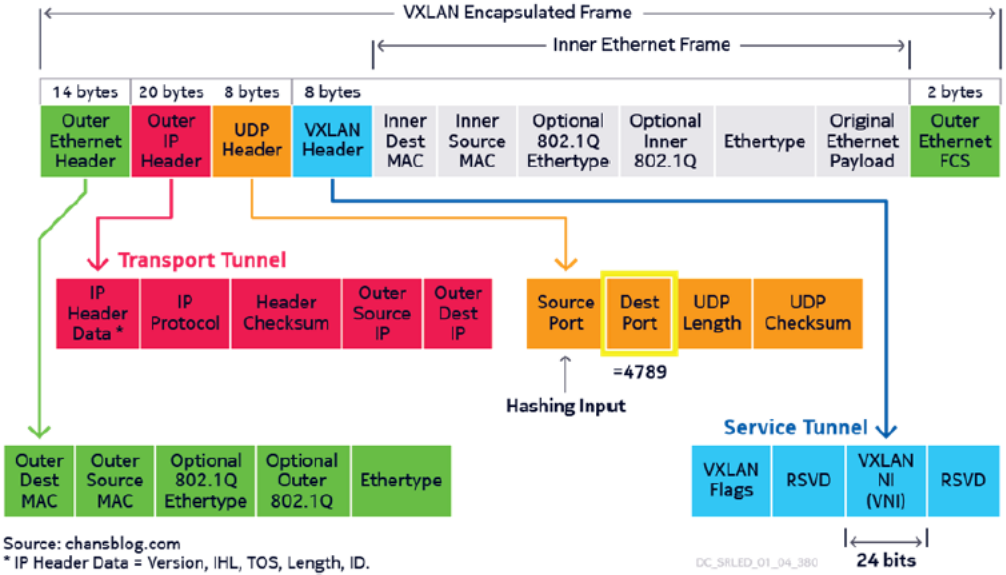
- UDP-based (port 4789)
- UDP source port is a hash of the original payload MAC, IPs, and ports to provide flow-based load balancing entropy

8-byte VXLAN header:

- VXLAN Network Identifier 24-bits
- Allows for 16M overlays

Since VXLAN is IP-based, it can be routed over any IP network:

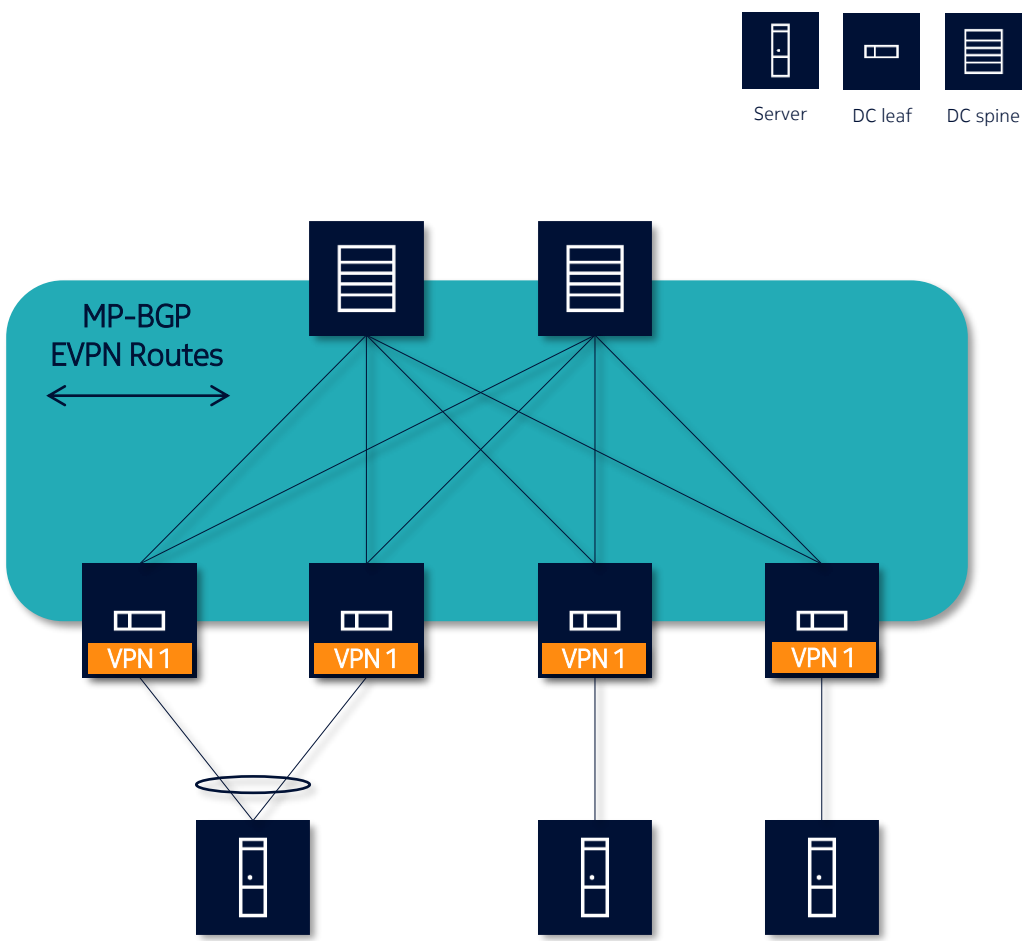
- Enable ECMP for load-balancing
- Network must support the 50byte encapsulation overhead



Control Plane

EVPN in data centers

- EVPN relies **on MP-BGP** in the service provider network, i.e. the data center fabric.
- Several EVPN route types are defined:
  - EVPN routes are **exchanged between leafs**, via a route reflector or not.
  - Route types advertised are based on the **use case** and the **type of service** being delivered.



Control Plane – EVPN Route types

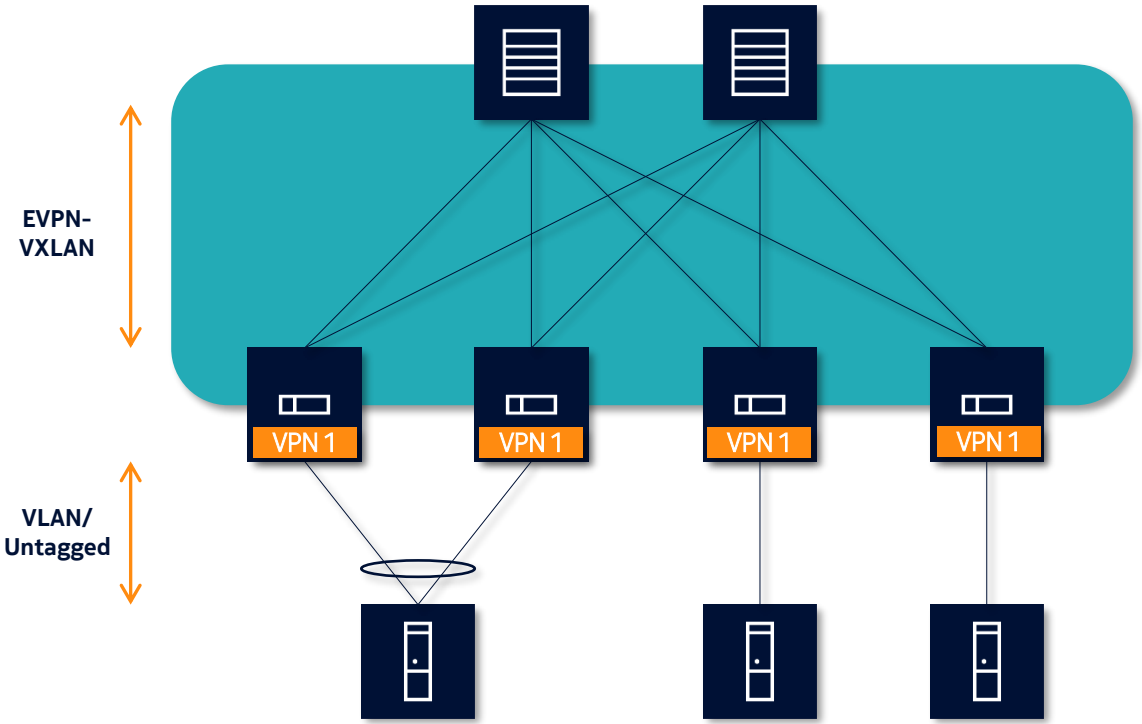
EVPN in data centers

Route type	Route name	Purpose
1	Ethernet Auto-Discover (A-D)	Used in multi-homing scenarios to support aliasing and fast convergence
2	**MAC/IP Advertisement**	**Used to advertise host MAC address or host MAC/IP addresses**
3	**Inclusive Multicast Ethernet Tag (IMET)**	**Used to discover member Pes and to setup the flooding tree for BUM traffic**
4	Ethernet Segment (ES)	Used in multi-homing scenarios to suppport Ethernet segment discovery and DF election
5	**IP-Prefix**	**Used to advertise IP prefixes for inter-subnet connectivity in L3VPN services**

Data Plane

EVPN in data centers

- EVPN allows the delivery of multiple services over a single core network
- Leafs encapsulates customer data with a **label that uniquely identifies each service** Encapsulated data is **tunnelled between leafs** (or border leafs, or DCGWs).
- **VXLAN** tunnels are used within the fabric.
- **MPLS** tunnels are used within IP/MPLS networks.

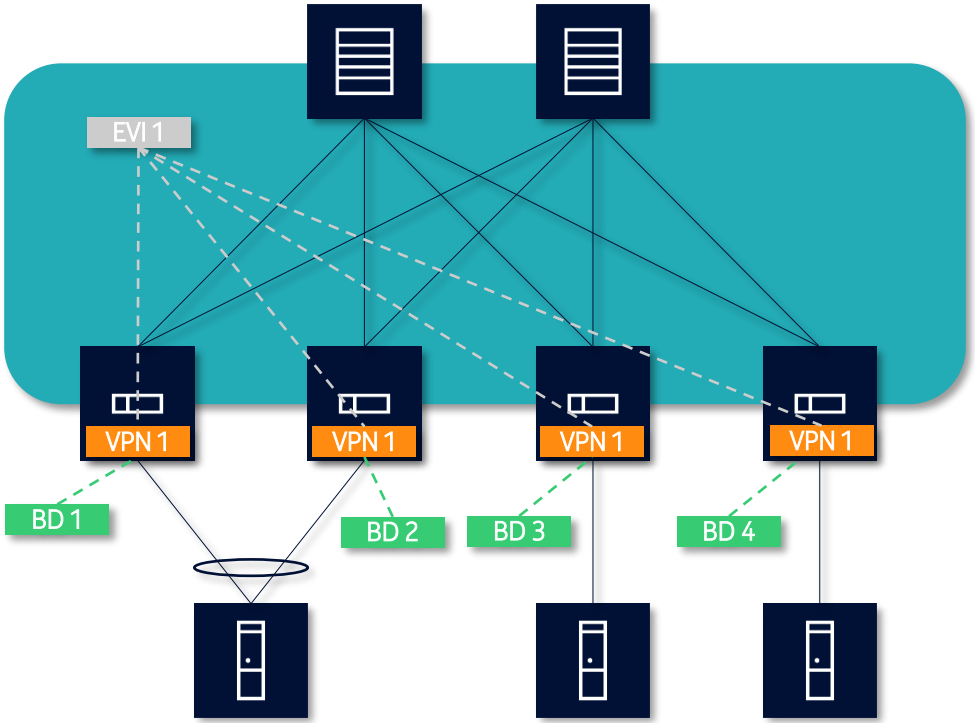


Layer-2 services - terminology

EVPN in data centers

- **Broadcast domain (BD)**
  - An instantiation of an EVPN service on a given leaf
- **MAC-VRF**
  - The virtual routing and forwarding (VRF) table that contains the MAC addresses for an EVPN service
- **EVPN instance (EVI)**
  - The group of BDs that are part of the same EVPN service

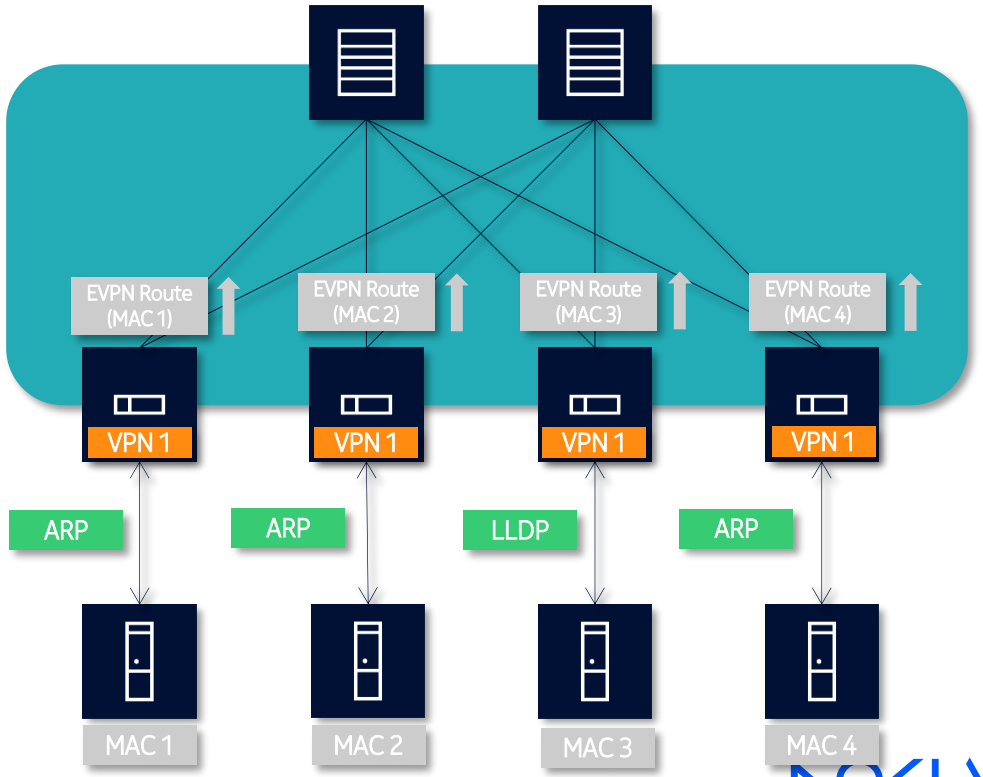
* In VLAN-aware bundle services, EVPN instances can consist of multiple broadcast domains. **On SR Linux, a broadcast domain instance on a leaf node is identified by a MAC-VRF, and a MAC-VRF can contain only one broadcast domain**. However, SR Linux supports an interoperability mode so that SR Linux leaf nodes can be attached to VLAN-aware bundle broadcast domains along with other third-party routers.



Layer-2 services – EVPN operation

EVPN in data centers

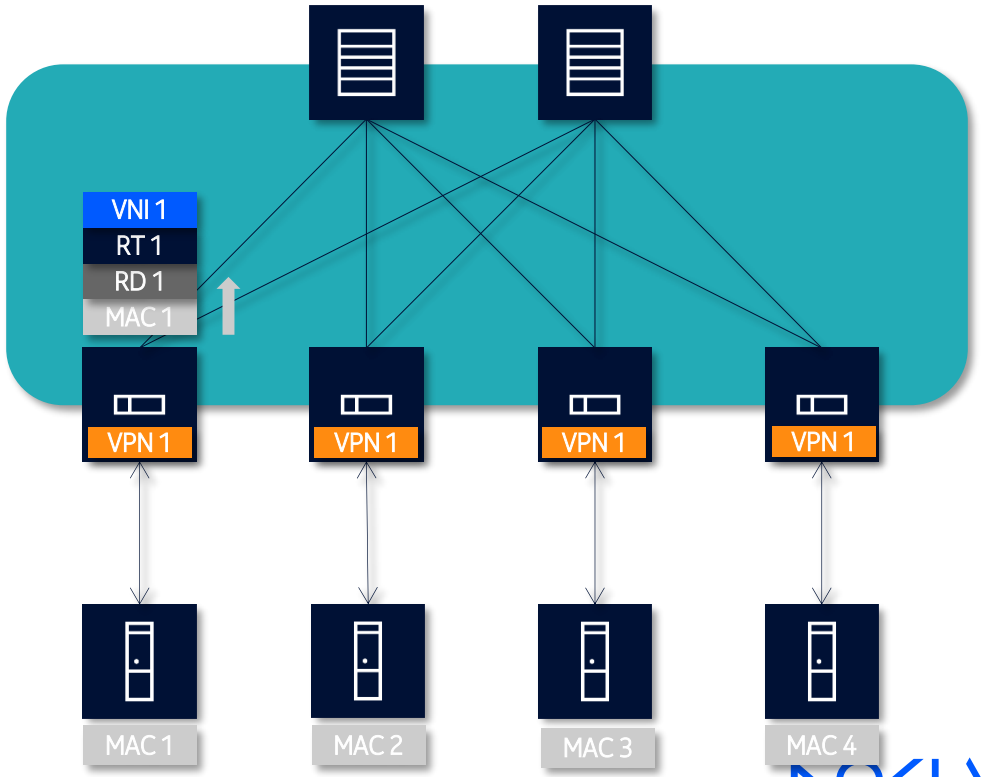
- EVPN enables control-plane MAC learning in the core
- Leafs exchange EVPN routes over **MP-BGP** to **advertise local MAC addresses** across the fabric
- Attached clients can be physical or virtual
- Leafs learn MAC addresses of locally-connected clients using data-plane learning, static provisioning or control-plane protocols.



Layer-2 services – Leaf-to-leaf MAC address advertisement

EVPN in data centers

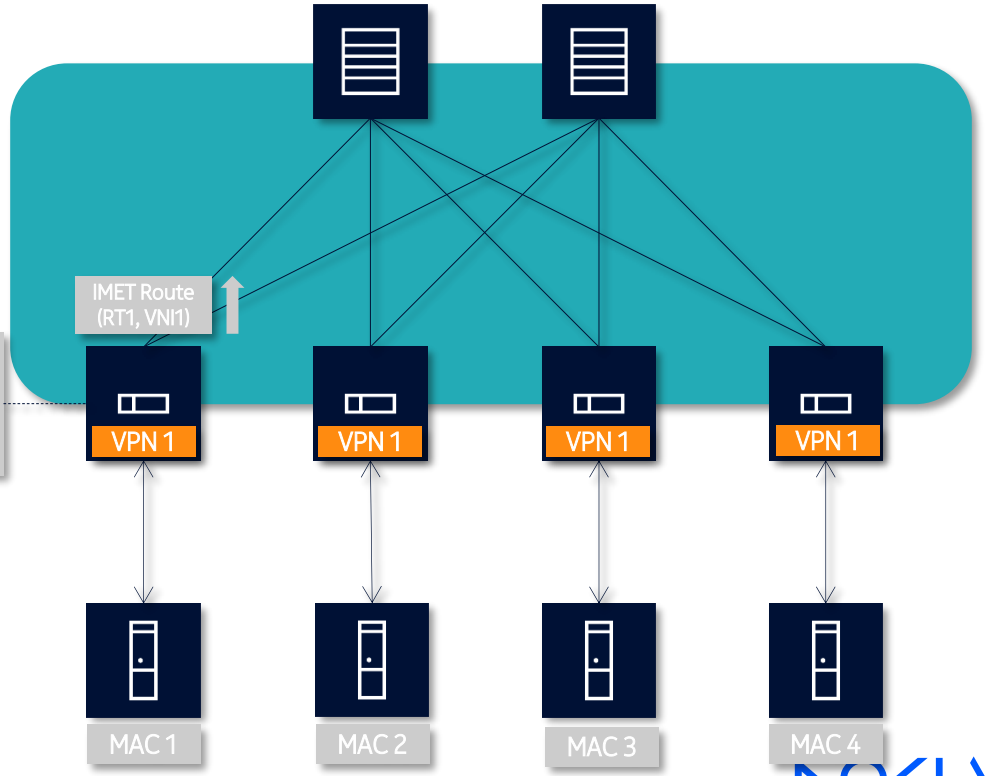
- Leafs advertise **locally-learned MAC addresses** using **MAC/IP routes** (EVPN route type 2)
- A single MP-BGP instance handles the exchange of routes for all EVIs on the leaf
- **Route distinguisher** is used to distinguish routes between EVIs in case of overlaps. **Route targets** identify which EVPN routes are to be installed in the local MAC-VRF
- Provided label (i.e. **VxLAN network identifier**) is used by remote leafs when encapsulating frames destined to the advertised MAC address



Layer-2 services – Flooding lists & BUM traffic handling

EVPN in data centers

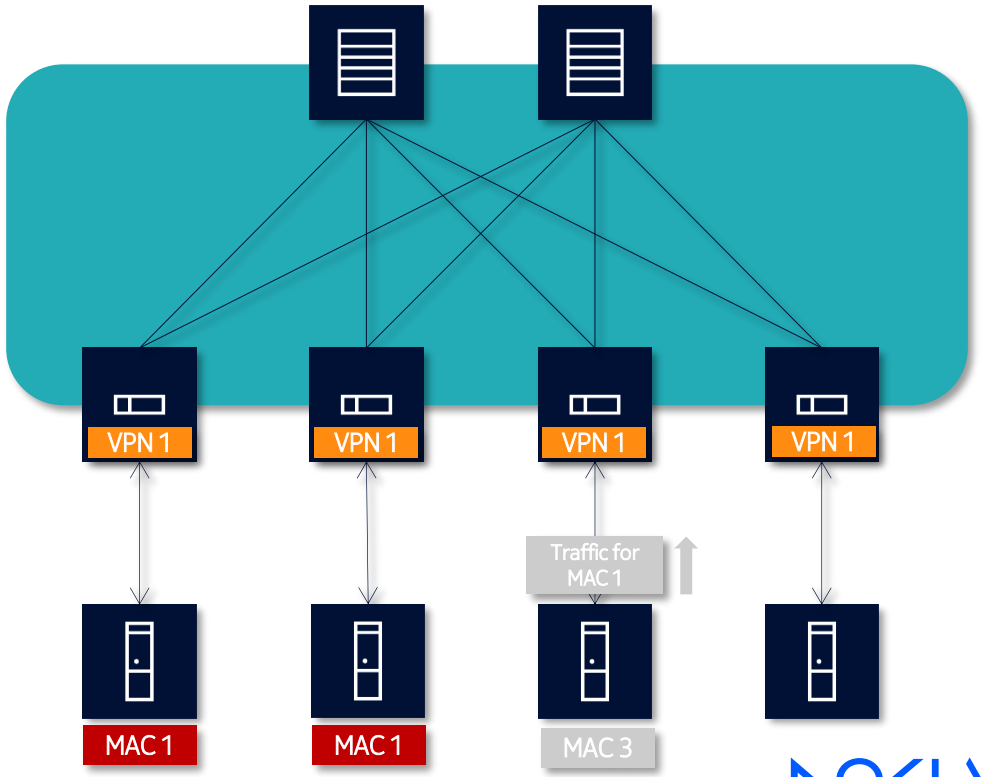
- When an EVPN service is enabled, leafs participating to this service exchange **inclusive multicast Ethernet tag routes** (**IMET** or type 3)
  - Discover all leafs attached to the same EVI
  - Build a flooding list in each leaf
- A leaf receiving a **BUM frame** from a client, **consults the flooding list** of the EVPN service to determine the leafs to which it needs to flood the frame
- A leaf receives this encapsulated BUM frame from the fabric, **decapsulates** the packet and then **floods it to its local interfaces**
  - Thanks to **split-horizon**, the BUM frame are never flooded back to the fabric



Layer-2 services – Dealing with layer 2 loops

EVPN in data centers

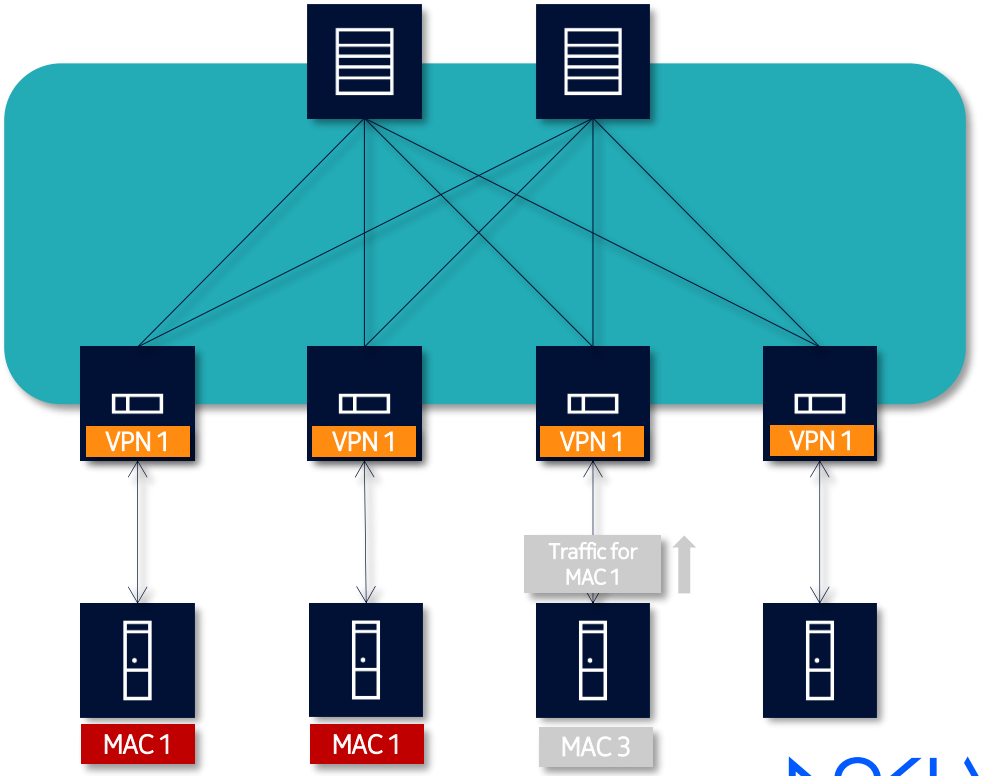
- The detection of duplicate MAC addresses and loops is a fundamental feature when extending a broadcast domain
- L2 loops or duplicate MACs are typically due to a configuration mistake or an intended spoofing attack.
- **MAC duplication** is the mechanism used by SR Linux for **loop prevention**. MAC duplication **monitors MAC addresses that move between subinterfaces**. It consists of detection, actions, and process restart.



Layer-2 services – MAC-duplication as loop protection mechanism

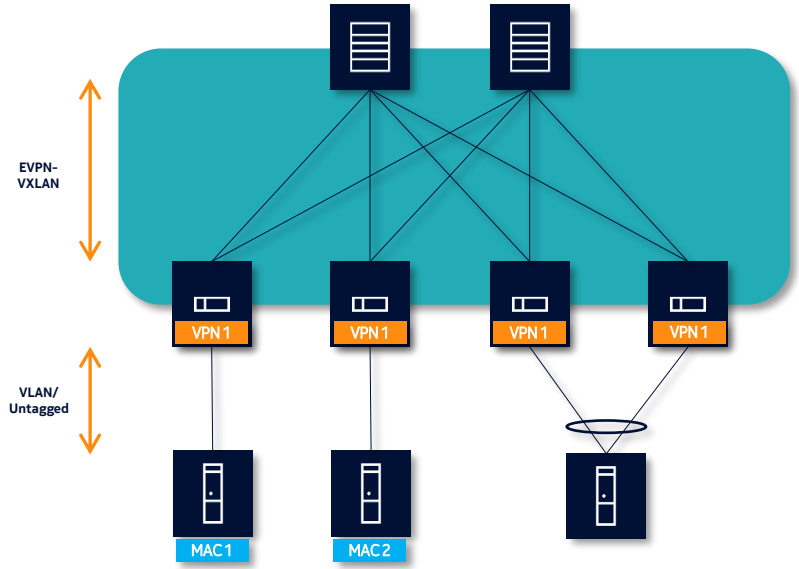
EVPN in data centers

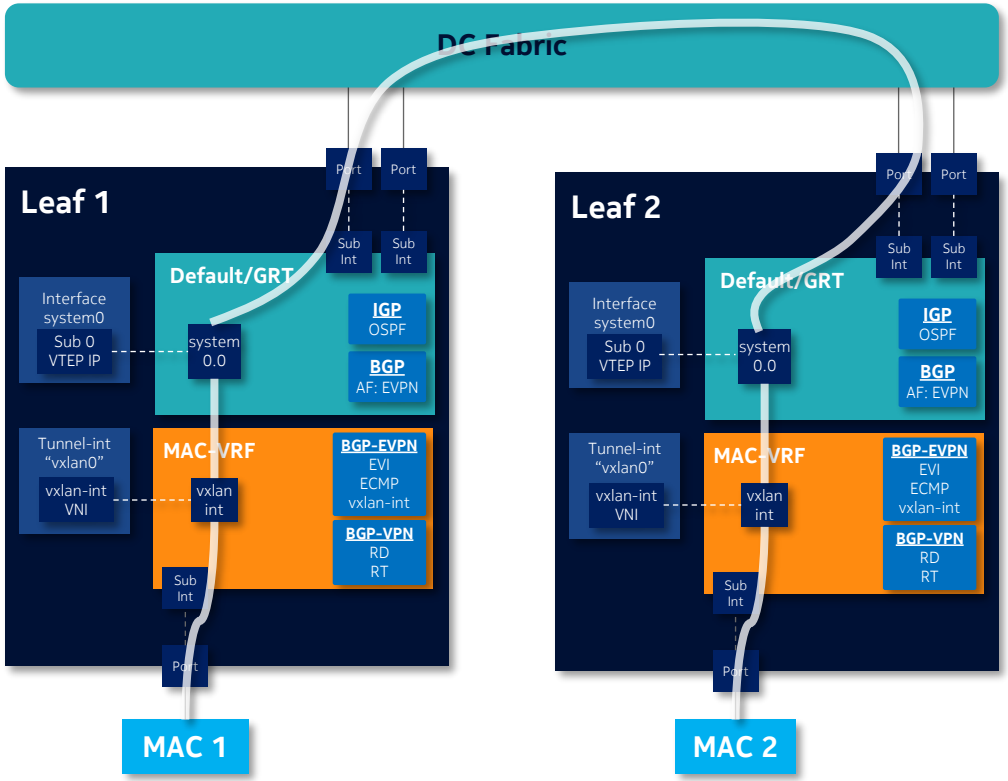
- A MAC is declared **duplicate** if it is learnt on **different interfaces** and **the number of moves is higher than a certain value** (num-moves) within a certain interval (monitoring-window).
- A **configurable action** can be performed on the subinterface when a duplicate MAC is detected. One of three options can be selected (stop-learning, blackhole, oper-down).
- The **MAC remains "duplicate"** for the duration of the **hold-down-time** parameter. At the end of that interval, it is flushed from the bridge table and the action on the subinterface is cleared.



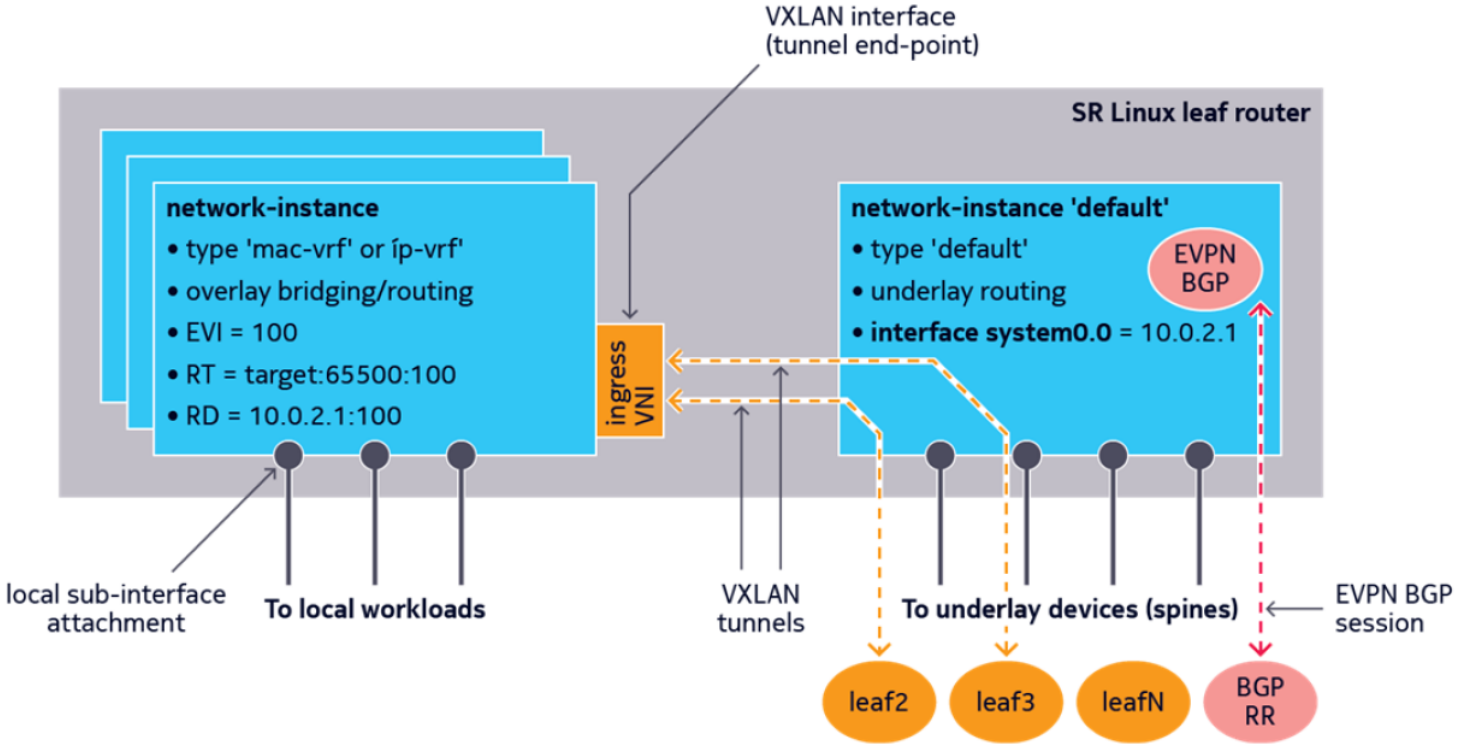
Configuring an EVPN layer-2 service on SR Linux

EVPN in data centers



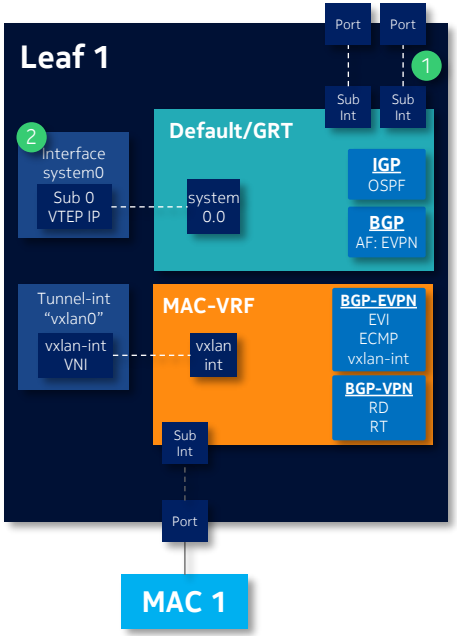


SR Linux EVPN – VXLAN Configuration Overview



Configuring an EVPN layer-2 service on SR Linux

EVPN in data centers



1

```
# info interface ethernet-1/55
    interface ethernet-1/55 {
        admin-state enable
        vlan-tagging true
        subinterface 1 {
            ipv4 {
                address 101.1.1.0/31 {
                }
            }
            ipv6 {
                address 2002::101:1:1:0/127 {
                }
            }
            vlan {
                encap {
                    single-tagged {
                        vlan-id 1
                    }
```

ethernet-1/55 is an uplink interface, i.e. towards the fabric

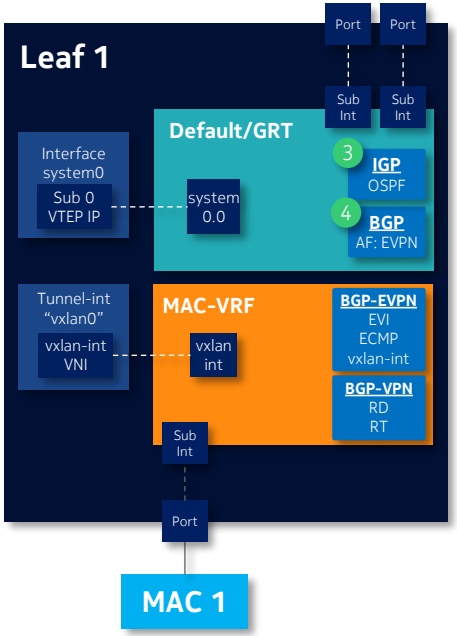
system0.0 is the loopback interface used to originate and terminate VxLAN packets

2

```
# info interface system0
    interface system0 {
        admin-state enable
        subinterface 0 {
            admin-state enable
            ipv4 {
                address 192.1.1.1/32 {
                }
            }
            ipv6 {
                address 2000::192:1:1:1/128 {
                }
            }
        }
    }
```

Configuring an EVPN layer-2 service on SR Linux

EVPN in data centers



3

```
# info network-instance default protocols ospf
    network-instance default {
        protocols {
            ospf {
                instance default {
                    admin-state enable
                    version ospf-v2
                    router-id 192.1.1.1
                    area 0.0.0.0 {
                        advertise-router-capability true
                        interface ethernet-1/55.1 {
                            interface-type point-to-point
                        }
                        interface ethernet-1/56.1 {
                            interface-type point-to-point
                        }
                        interface system0.0 {
                        }
```

OSPF is chosen as underlay protocol in this example, but **IS-IS** or **eBGP (preferred)** are also supported.

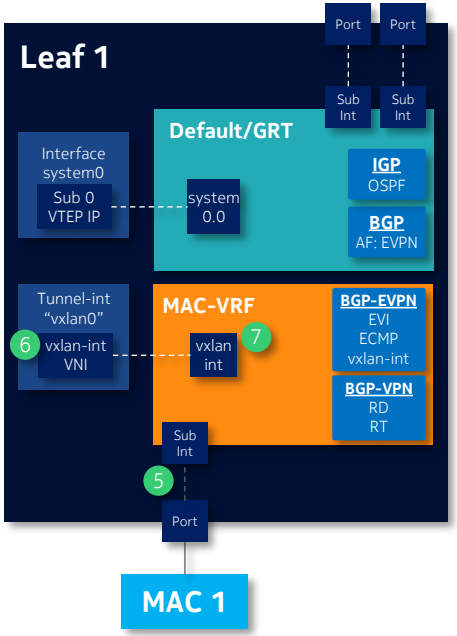
iBGP with address family EVPN is used to exchange the EVPN routes between the different VTEPs.

4

```
# info network-instance default protocols bgp
    network-instance default {
        protocols {
            bgp {
                autonomous-system 64500
                router-id 192.1.1.1
                group iBGPv4 {
                    admin-state enable
                    peer-as 64500
                    ipv4-unicast {
                        admin-state disable
                    }
                    ipv6-unicast {
                        admin-state disable
                    }
                    evpn {
                        admin-state enable
                    }
                    timers {
                        connect-retry 1
                        minimum-advertisement-interval 1
                    }
                }
                neighbor 192.1.2.1 {
                    peer-group iBGPv4
                }
                neighbor 192.1.2.2 {
                    peer-group iBGPv4
                }
```

Configuring an EVPN layer-2 service on SR Linux

EVPN in data centers



5

```
# info interface ethernet-1/3 subinterface 110
    interface ethernet-1/3 {
                vlan-tagging true
        subinterface 110 {
            type bridged
            admin-state enable
            vlan {
                encap {
                    single-tagged {
                        vlan-id 110
                    }
```

6

```
# info tunnel-interface vxlan0
    tunnel-interface vxlan0 {
        vxlan-interface 110 {
            type bridged
            ingress {
                vni 110
            }
            egress {
                source-ip use-system-ipv4-address
            }
        }
    }
```

7

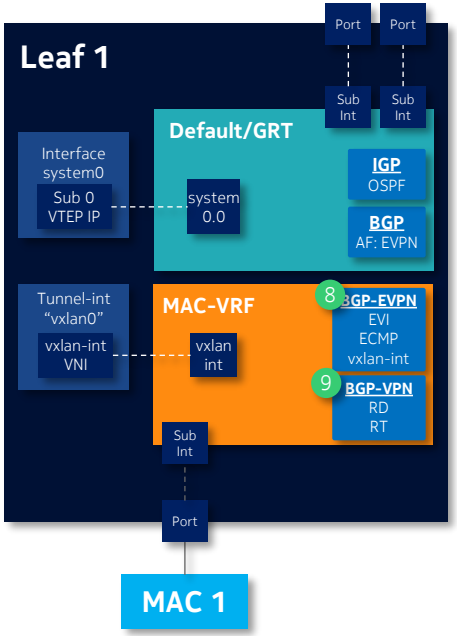
```
# info network-instance mac-vrf-110
    network-instance mac-vrf-110 {
        type mac-vrf
        admin-state enable
        description "Simple EVPN Layer 2"
        interface ethernet-1/3.110 {
        }
        vxlan-interface vxlan0.110 {
        }
```

Possible options:

- **single-tagged vlan-id any** – where 'any' captures all traffic for which no specific vlan-id has been defined.
- **untagged** – where 'untagged' captures traffic with no tags or vlan-tag 0.

Configuring an EVPN layer-2 service on SR Linux

EVPN in data centers



8

```
# info network-instance mac-vrf-110 protocols bgp-evpn
    network-instance mac-vrf-110 {
        protocols {
            bgp-evpn {
                bgp-instance 1 {
                    admin-state enable
                    vxlan-interface vxlan0.110
                    evi 110
                    ecmp 2
                    routes {
                        bridge-table {
                            next-hop use-system-ipv4-address
                            mac-ip {
                                advertise true
                            }
                            inclusive-mcast {
                                advertise true
                            }
```

9

```
# info network-instance mac-vrf-110 protocols bgp-vpn
    network-instance mac-vrf-110 {
        protocols {
            bgp-vpn {
                bgp-instance 1 {
                    route-distinguisher {
                        rd 110:11
                    }
                    route-target {
                        export-rt target:64500:110
                        import-rt target:64500:110
                    }
```

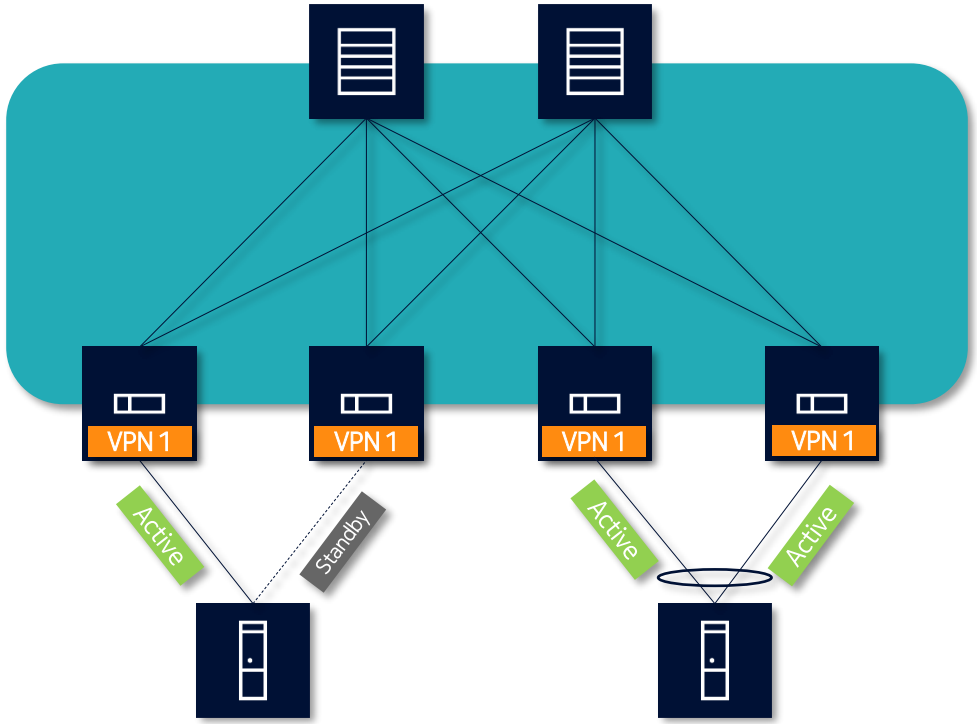
- RD can be auto-derived from EVI if not configured manually as <system-ip:evi>
- RT can be auto-derived from EVI if not configured manually as <AS:evi>

Multi-homing - terminology

EVPN in data centers

- **Single-active** mode
  - Multi-homed with one-active leaf at any time. A single leaf forwards traffic to and from the client

- **All-active** mode
  - Multi-homed with **two or more active leafs** (up to 4 with SR Linux and 7750 SR)
  - **LAG is required** on the client side, to avoid duplicate packets and forwarding loops



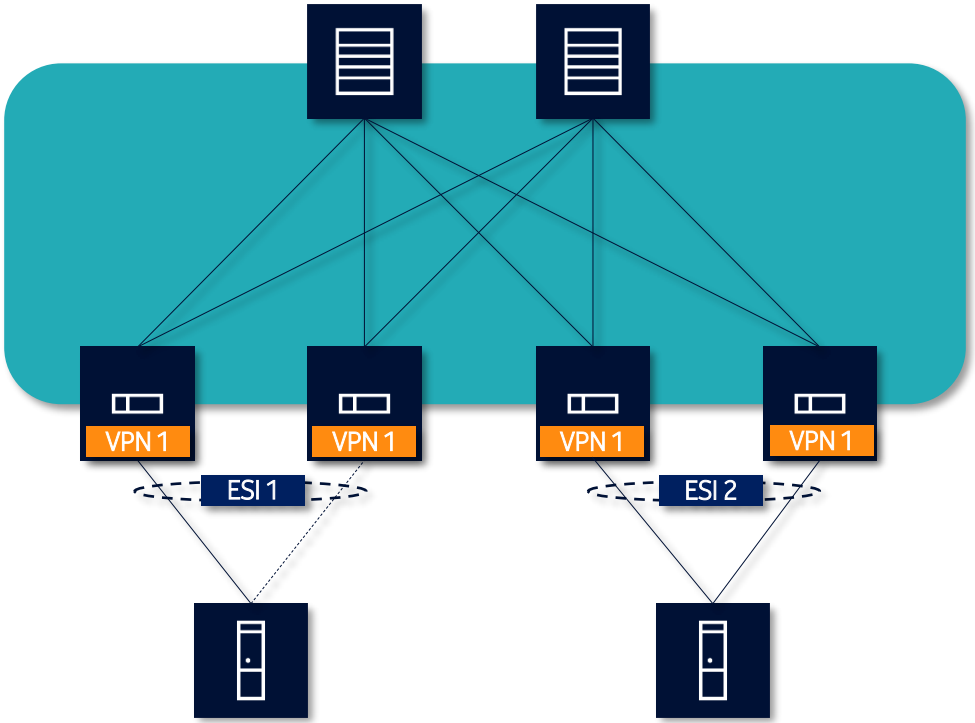
Multi-homing - terminology

EVPN in data centers

Server DC leaf DC spine

Multi-homing in EVPN is based on the concept of Ethernet Segment.

- **Ethernet Segment** (ES)
  - Represents a set of links that connect a client to one or more leafs
  - In single-active or all-active mode
  - On leafs, an ES consists of physical or logical links (LAG, port, vlan ID, …)
- **ES identifier** (ESI)
  - Uniquely identifies an ES in the fabric
  - ESI 0 – indicates a single-homed site
  - ESI 0xFF – reserved, Max-ESI



Multi-homing – EVPN Routes

EVPN in data centers

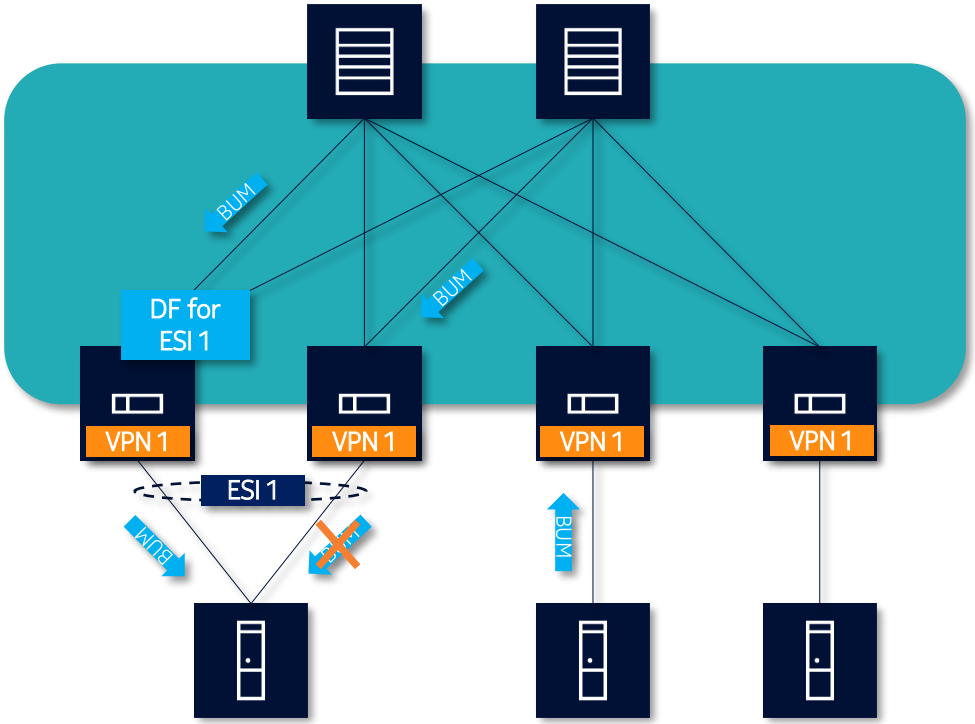
When it comes to multi-homing, leafs exchange **two important route types** between each other :

- Routes of type 1 Ethernet Auto-Discovery come in two flavours :
  - **A-D per ES** : used to **discover ES** and identify the list of leafs associated with an ES.
    It indicates the **ES redundancy mode** (all-active or single-active), and includes the **ESI label** required for split-horizon. Also used for mass withdrawal.
  - **A-D per EVI** : used to **advertise the ES availability in a given EVI**. It's mainly used to create aliasing lists and to create primary/backup lists of leafs that are part of a single-active ES.
- Routes of type 4 Ethernet Segment
  - Used to **discover leafs** attached to a given ES and to **elect a designated forwarder**.
  - A leaf advertises an **ES route for each locally provisioned ES** that has an operational service associated with it.
  - ES routes are advertised with a special RT derived from the ESI
    - Leafs that are part of the ES will import the route; if not, they won't.

Multi-homing – handling BUM traffic to All-Active clients

EVPN in data centers

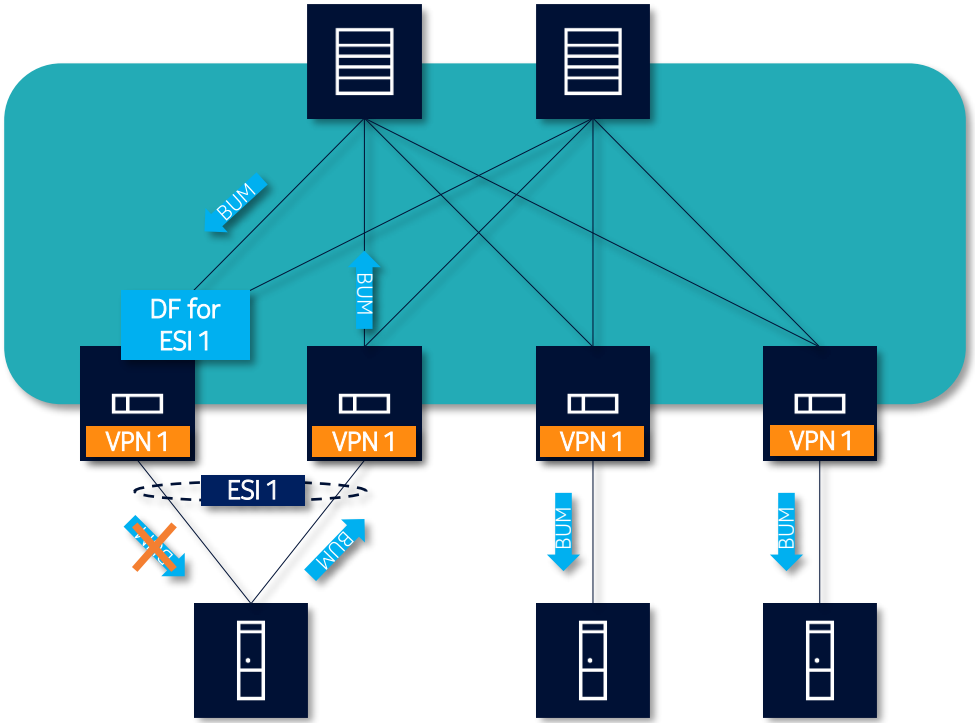
- Leafs connected to a multi-homed client discover each other
- One leaf is elected as Designated Forwarder (DF) per ESI
- Only the DF leaf floods BUM traffic to the Ethernet Segment.



Multi-homing – handling BUM traffic from All-Active clients

EVPN in data centers

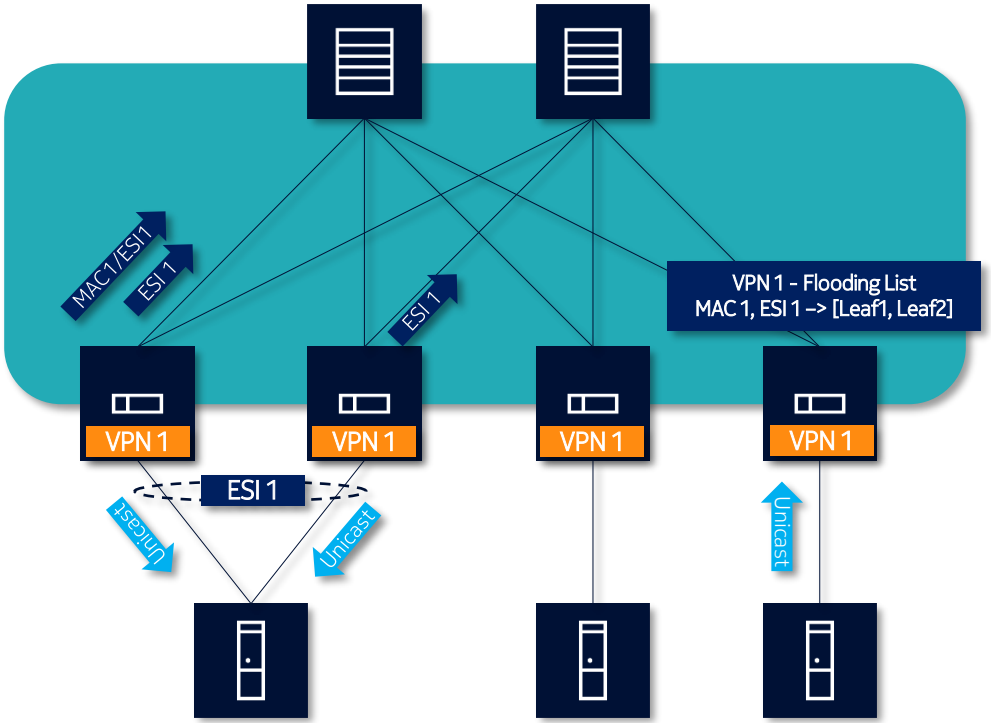
- Ingress leaf **encapsulates the BUM packet with an ESI label** and sends the packet to each member of its flooding list.
  - ESI label identifies the originating ES
- Egress leaf **does not forward packet** to the ES identified by the ESI label **if it is connected to that same ES**. This avoids replication of the traffic originated from an ES back to that same ES. Also called split-horizon, in EVPN, this specific mechanism is called **local bias**.



Multi-homing – aliasing for All-Active clients

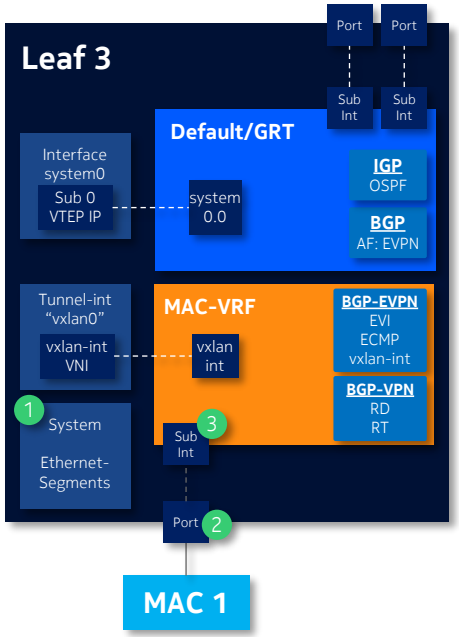
EVPN in data centers

- Leafs advertise their **local ESIs** in **Ethernet Auto-Discovery Routes** (EVPN Route Type 1)
- MAC/IP routes identify the ES of the advertised MAC, in the MAC-IP Advertisement Routes (EVPN Route Type 2)
- When sending unicast traffic, this list allows **load-balancing** to all the ES peers attached to that EVI. This mechanism is called **aliasing**.



MH - Configuring an EVPN layer-2 service on SR Linux

EVPN in data centers



1

```
A:leaf14# info system network-instance protocols evpn ethernet-segments bgp-instance 1 ethernet-segment ES-120
  system {
    network-instance {
      protocols {
        evpn {
          ethernet-segments {
            bgp-instance 1 {
                                        ethernet-segment ES-2 {
                                                admin-state enable
                                                esi 01:01:00:00:00:00:00:00:00:02
                                                multi-homing-mode all-active
                                                interface lag2 {
```

2

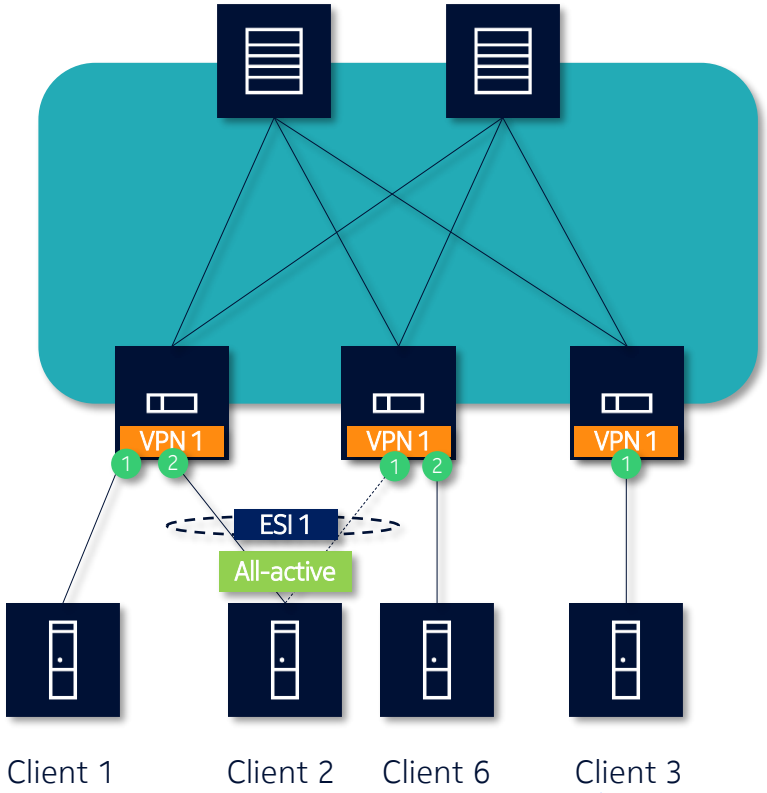
```
A:leaf14# info interface lag2
  interface lag2 {
    admin-state enable
    vlan-tagging true
    subinterface 120 {
      type bridged
      vlan {
        encap {
          single-tagged {
            vlan-id 120
    lag {
      lag-type lacp
      member-speed 10G
      lacp {
        interval FAST
        lacp-mode ACTIVE
        admin-key 2
        system-id-mac 00:00:00:00:00:02
```

3

```
A:leaf14# info network-instance mac-vrf-120
  network-instance mac-vrf-120 {
    type mac-vrf
    admin-state enable
    interface lag2.120 {
    }
```

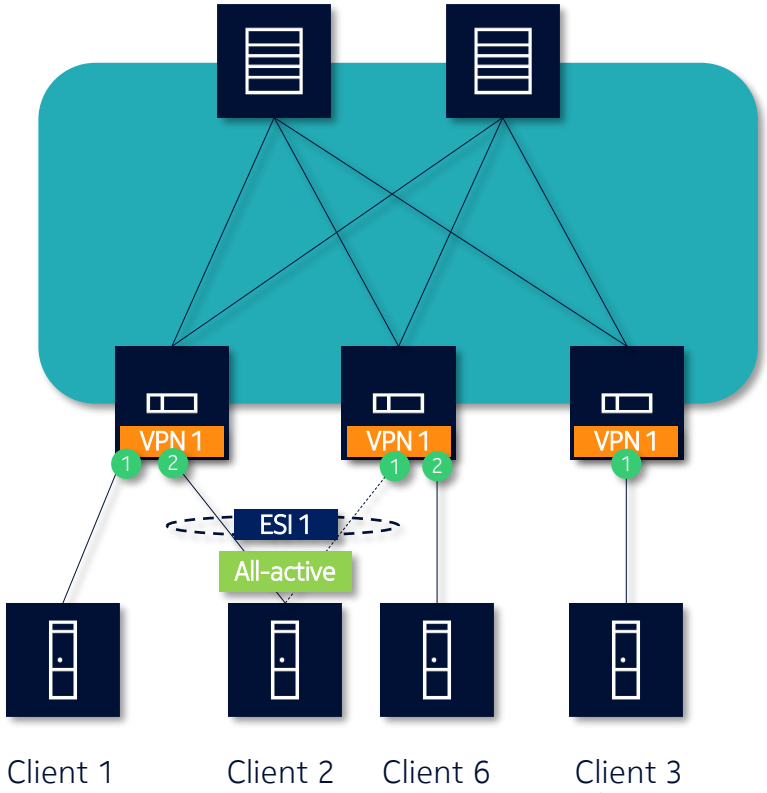
Hands-on activity #2

- The previous activity deployed the underlay and an EVPN control plane.
- Let's now configure an overlay and subinterfaces to connected clients. With the help of the previous slides, achieve the following scenarios.
- Task 1 – **Single-homed** clients:
  - Configure subinterfaces on customer-facing ports
  - Create a vxlan interface
  - Create a MAC-VRF on Leaf 1, 2 and 3
  - Associate relevant subinterfaces to the MAC-VRF
- Task 2 – **Multi-homed** client:
  - Define a LAG interface on leaf 1 and 2 and make sure ports facing client 2 are included in it.
  - Define an Ethernet Segment on leaf 1 and 2.
- For both tasks, verify that the correct routes are being exchanged over EVPN, and simulate a traffic test (see next page)



Hands-on activity #2 – traffic simulation

- If your leafs are correctly configured, you should be able to run traffic in your fabric!
- A script is in your group directory:
  - /home/group**X**/innog8-workshop/day_2-ixp-dc-lab/traffic.sh
  - The script triggers an iperf that generates traffic between different clients.
  - To start all the pre-programmed flows, type:
    - ./traffic.sh start all
  - To stop the traffic, type:
    - ./traffic.sh stop all
  - While sending traffic, observe on your Grafana dashboard that traffic is flowing through the fabric and received correctly.
    - http://45.76.181.43:300**X**/




NOKIA

Copyright and confidentiality