Hands-On SR-MPLS

Ivan Pepelnjak (ip@ipSpace.net)
Network Architect

ipSpace.net AG

Who is Ivan Pepelnjak (@ioshints)

Past

- Kernel programmer, network OS, and web developer
- Sysadmin, database admin, network engineer, CCIE
- Trainer, course developer, curriculum architect
- Team lead, CTO, business owner
- SDN skeptic and network automation evangelist

Present

- Network architect, consultant, blogger, open-source developer

Focus

- SDN and network automation
- Large-scale data centers, clouds, and network virtualization
- Scalable application design
- Core IP routing/MPLS, IPv6, VPN

CISCO CCIE EMERITUS

vmware vEXPERT

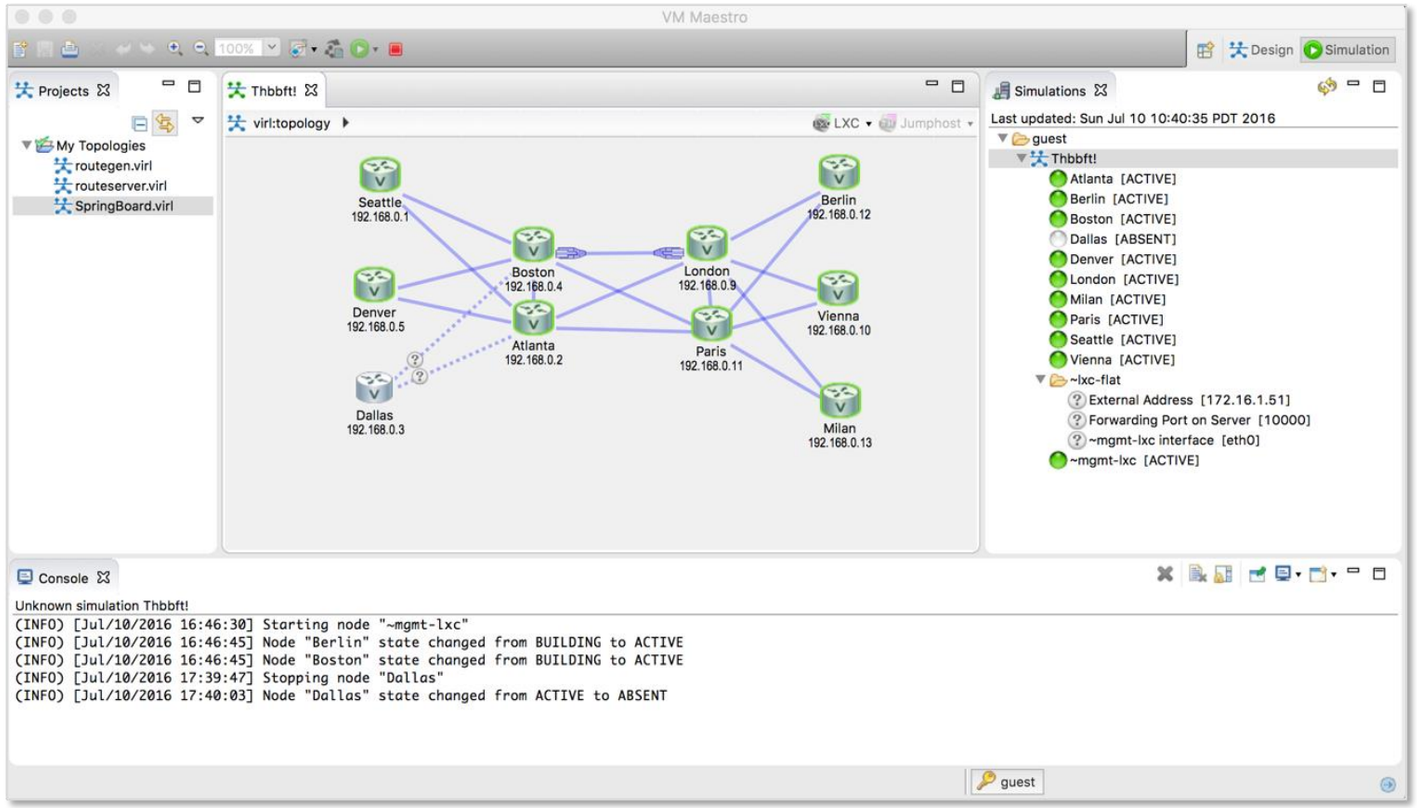
Also: I built too many labs for one lifetime, and hated that with passion


That SR Thing Sounds Cool, But Does It Work?


LET'S ASK CHATGPT


Today's Side Quest
Make Network Labs Fun
Let's Try It Out in a Lab

Everything Is Better with a Digital Twin

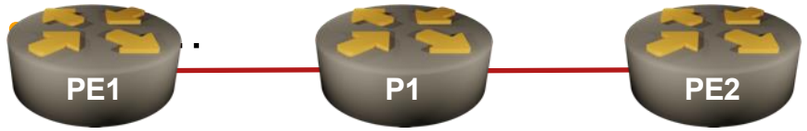


Did you ever want to:

- Export or share your topologies?
- Have a quick way to modify them?
- Use version control to rollback bloopers?

What we need is Infrastructure-as-Code solution for network labs

Infrastructure-As-Code Labs



Early solutions

- Vagrant (virtual machines)
- Docker Compose (containers)

Clear winner: containerlab

- Networking focus
- Native container or virtual machines in containers (vrnetlab)
- Builds point-to-point links between containers (no Linux bridge shenanigans)

containerlab.yml

```
topology:
  kinds:
    ceos:
      env: {'CLAB_MGMT_VRF': 'management'}
      image: ceos:4.35.2F
  nodes:
    pe1:
      kind: ceos
    p:
      kind: ceos
    pe2:
      kind: ceos

  links:
  - endpoints:
    - "pe1:et1"
    - "p:et1"
  - endpoints:
    - "p:et2"
    - "pe2:et1"
```


Demo time: Start a lab with containerlab

A Networking Lab Is Much More than Topology

- Build the lab topology
- Create addressing plan
- Initial device configurations
- Configure IS-IS (or OSPFv2)
- Add SR-MPLS configuration


What If We Would have a Magic Tool...

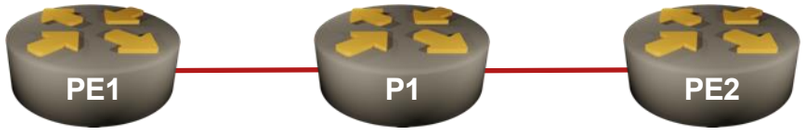
In the Unicorn Land of Infrastructure-as-Code Intent-Based Labs

Create a high-level description of the network

- Three devices
- … they are Arista EOS containers
- … running IS-IS and SR-MPLS
- We also need two links

Next

- Save the file
- Execute **netlab up** and you'll get a running network (including IP addressing, IS-IS and SR-MPLS)

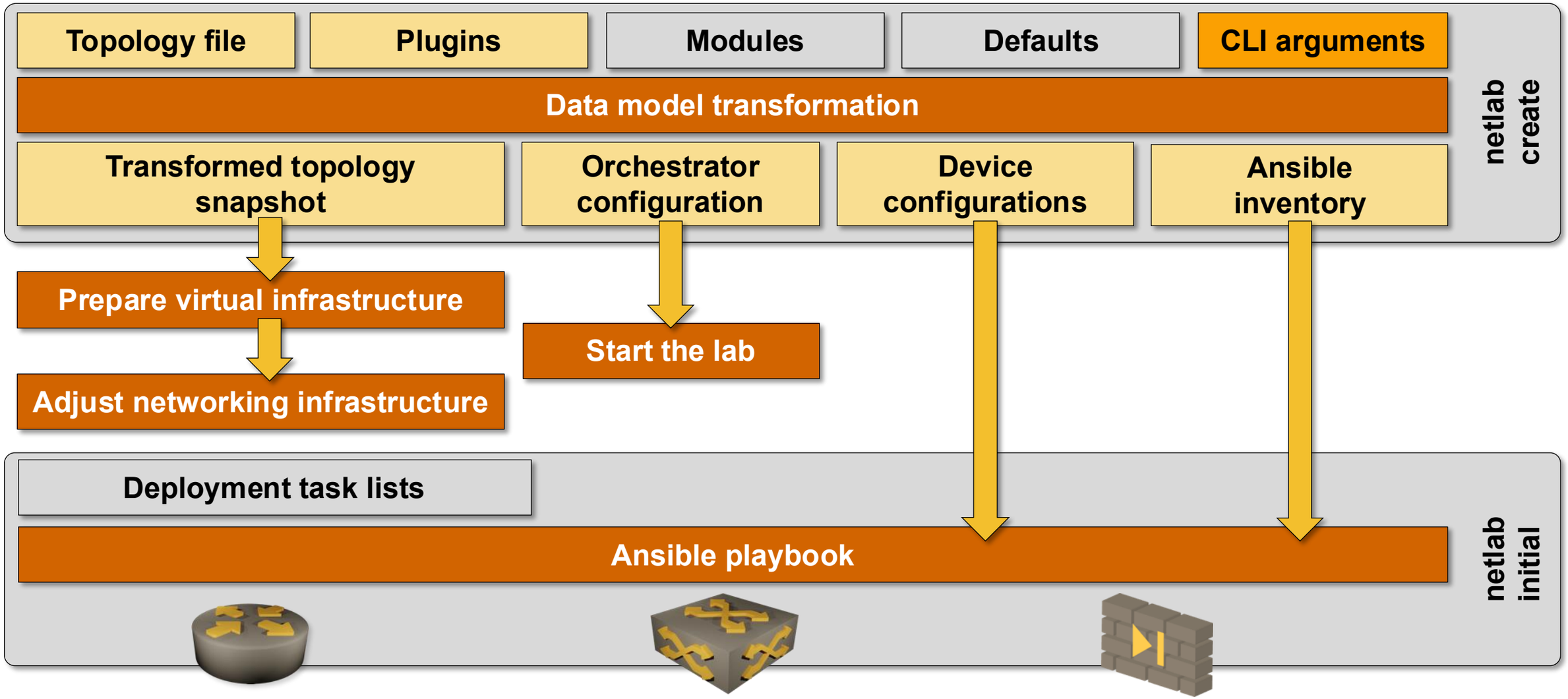


topology.yml

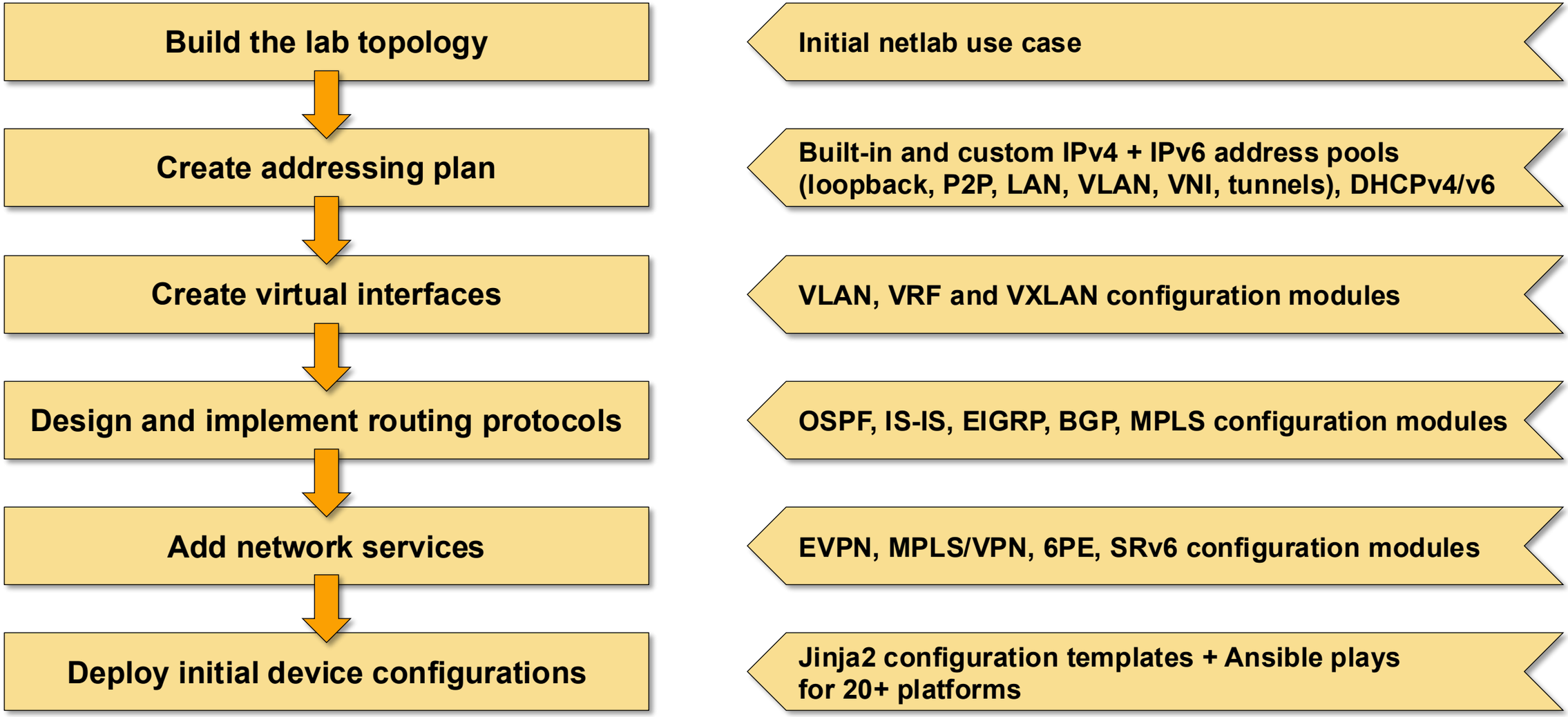
```
nodes: [ pe1, p, pe2 ]

defaults.device: eos
provider: clab

module: [ isis, sr ]

links: [ pe1-p, p-pe2 ]
```


Demo time: Start a lab with netlab

Wait, What Just Happened? netlab up Behind the Scenes



Building a Networking Lab with netlab



Can I Use My Favorite Device?

The following table describes per-platform support of individual IS-IS features:

Operating system	IS type	Circuit type	IPv6 AF	Multi topology
Arista EOS	✅	✅	✅	✅
Cisco ASAv	✅	✅	✅	✅
Cisco IOS/XE[1]	✅	✅	✅	✅
Cisco IOS XR[2]	✅	✅	✅	✅
Cisco Nexus OS	✅	✅	✅	✅
FRR	✅	✅	✅	✅
Junos[3]	✅	✅	✅	✅
Nokia SR Linux	✅	✅	✅	✅
Nokia SR OS[4]	✅	✅	✅	✅
VyOS	✅	✅	✅	✅

SR-MPLS is implemented on the following platforms:

Operating system	IPv4	IPv6	IS-IS	OSPFv2
Arista EOS	✅	✅	✅	✅
Cisco IOS XE[1]	✅	❌	✅	✅
Cisco IOS XR[2]	✅	✅	✅	✅
FRRouting	✅	✅	✅	✅
Junos[3]	✅	✅	✅	❌
Nokia SR Linux !	✅	❌	✅	❌
Nokia SR OS[4]	✅	✅	✅	❌

More @ https://netlab.tools/platforms/

Recap: Lab Addressing

```
$ netlab report addressing
[INFO]    Using lab topology file topology.yml
  Node/Interface           IPv4 address                                  IPv6 address Description
=================================================================================================
pe1 (10.0.0.1/32)
  Ethernet1                10.1.0.2/30                                                pe1 -> p

p (10.0.0.2/32)
  Ethernet1                10.1.0.1/30                                                p -> pe1
  Ethernet2                10.1.0.5/30                                                p -> pe2

pe2 (10.0.0.3/32)
  Ethernet1                10.1.0.6/30                                                pe2 -> p

$
```

Recap: IS-IS Parameters

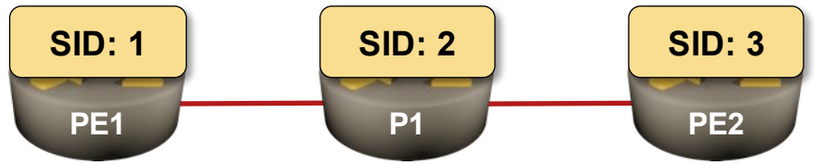
```
$ netlab report isis-nodes
[INFO]    Using lab topology file topology.yml
+------------------+-----------------+-------------+----------------+-----------+
| Node             | Router ID       | IS-IS area  | System ID      | IS Type   |
+------------------+-----------------+-------------+----------------+-----------+
| pe1              |        10.0.0.1 |      49.0001 | 0000.0000.0001 | level-2   |
| p                |        10.0.0.2 |      49.0001 | 0000.0000.0002 | level-2   |
| pe2              |        10.0.0.3 |      49.0001 | 0000.0000.0003 | level-2   |
+------------------+-----------------+-------------+----------------+-----------+

$
```

Recap: Inspecting the Status of SR-MPLS with IS-IS

```
pe1#show isis segment-routing

System ID: pe1                  Instance: Gandalf
SR supported Data plane: MPLS                   SR Router ID: 10.0.0.1
SR Global Block( SRGB ):
Base            Size
900000          65536
Total SRGB Size: 65536

Adj-SID allocation mode: SR-adjacencies
Adj-SID allocation pool: Base: 100000     Size: 16384

All Prefix Segments have    : P:0 E:0 V:0 L:0
IS-IS Reachability Algorithm : SPF (0)
Proxy-node segment attached flag: ignored

Number of IS-IS segment routing capable nodes excluding self: 2

Self-Originated Segment Statistics:
Node-Segments       : 1
Prefix-Segments     : 0
Proxy-Node-Segments : 0
Adjacency Segments  : 1
```

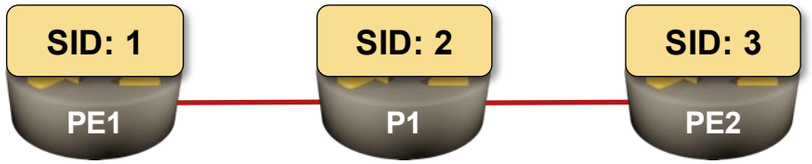
Recap: Inspecting Prefix Segments

```
pe1#show isis segment-routing prefix-segments

System ID: pe1                  Instance: 'Gandalf'
SR supported Data-plane: MPLS                   SR Router ID: 10.0.0.1

Node: 3      Proxy-Node: 0      Prefix: 0       Total Segments: 3

Flag Descriptions: R: Re-advertised, N: Node Segment, P: no-PHP
                   E: Explicit-NULL, V: Value, L: Local, A: Proxy-Node attached
Segment status codes: * - Self originated Prefix, L1 - level 1, L2 - level 2, ! - SR-unreachable,
                      # - Some IS-IS next-hops are SR-unreachable
   Prefix                   SID   Label Type       Flags                       System ID       Level Protection          Algorithm
   ------------------------ ----- ------- ---------- --------------------------- --------------- ----- ------------------- -------------
*  10.0.0.1/32               1   900001 Node       R:0 N:1 P:0 E:0 V:0 L:0     pe1             L2    unprotected         SPF
   10.0.0.2/32               2   900002 Node       R:0 N:1 P:0 E:0 V:0 L:0     p               L2    unprotected         SPF
   10.0.0.3/32               3   900003 Node       R:0 N:1 P:0 E:0 V:0 L:0     pe2             L2    unprotected         SPF
pe1#
```



Recap: Inspecting MPLS LFIB

```
pe1#show mpls route | begin 1000
 100000  A[1]
                via M, 10.1.0.1, pop
                    EgressACL: apply
                    directly connected, Ethernet1
                    ca:f0:00:02:00:01, vlan 1006
 900002  A[1]
                via M, 10.1.0.1, pop
                    EgressACL: apply
                    directly connected, Ethernet1
                    ca:f0:00:02:00:01, vlan 1006
 900003  A[1]
                via M, 10.1.0.1, swap 900003
                    EgressACL: apply
                    directly connected, Ethernet1
                    ca:f0:00:02:00:01, vlan 1006
pe1#
```

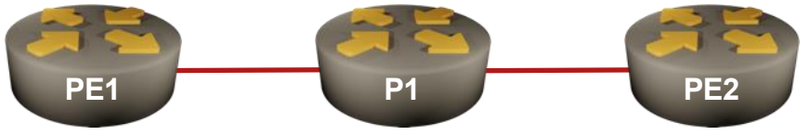


Recap: Inspecting SR-MPLS Adjacency Segments

```
pe1#show isis segment-routing adjacency-segments

System ID: pe1                  Instance: Gandalf
SR supported Data-plane: MPLS                   SR Router ID: 10.0.0.1
Adj-SID allocation mode: SR-adjacencies
Adj-SID allocation pool: Base: 100000     Size: 16384
Adjacency Segment Count: 1
Flag Descriptions: F: IPv6 address family, B: Backup, V: Value
                   L: Local, S: Set

Segment Status codes: L1 - Level-1 adjacency, L2 - Level-2 adjacency, P2P - Point-to-Point adjacency, LAN - Broadcast adjacency
                      ! - SR-unreachable, # - Some IS-IS next-hops are SR-unreachable

Locally Originated Adjacency Segments
   Adj IP Address       Local Intf          SID       SID Source                    Flags         Type     Protection
------------------- --------------- ------------ ---------------- ------------------------- ------------ -----------
        10.1.0.1             Et1       100000          Dynamic      F:0 B:0 V:1 L:1 S:0       P2P L2    unprotected

pe1#
```

Dual-Stack SR-MPLS

Dual-Stack SR-MPLS



Change address pools

- IPv4 and IPv6 loopbacks
- Core links with IPv4 and IPv6 addresses

Change other parameters

- Different IS-IS area
- Use FRRouting
- Define core links as a link group

topology.yml

```
provider: clab
defaults.device: frr
module: [ isis, sr ]

isis.area: "49.0042"
addressing:
  loopback:
    ipv4: 10.0.42.0/24
    ipv6: 2001:db8::/48
  core:
    ipv4: 192.168.42.0/24
    prefix: 31
    ipv6: 2001:db8:cafe::/48

nodes: [ pe1, p, pe2 ]
links:
- group: core
  pool: core
  members: [ pe1-p, p-pe2 ]
```


Demo time: Dual-Stack SR-MPLS

Recap: Lab Addressing

```
$ netlab report addressing
  Node/Interface           IPv4 address                                IPv6 address Description
=================================================================================================
pe1 (10.0.42.1/32 / 2001:db8:0:1::1/64)
  eth1                   192.168.42.1/31                          2001:db8:cafe::2/64 pe1 -> p

p (10.0.42.2/32 / 2001:db8:0:2::1/64)
  eth1                   192.168.42.0/31                          2001:db8:cafe::1/64 p -> pe1
  eth2                   192.168.42.2/31                        2001:db8:cafe:1::1/64 p -> pe2

pe2 (10.0.42.3/32 / 2001:db8:0:3::1/64)
  eth1                   192.168.42.3/31                        2001:db8:cafe:1::2/64 pe2 -> p
```

Recap: IS-IS Parameters

```
$ netlab report isis-nodes
+------------------+-----------------+--------------+-----------------+-----------+
| Node             | Router ID       | IS-IS area   | System ID       | IS Type   |
+------------------+-----------------+--------------+-----------------+-----------+
| pe1              |      10.0.42.1 |      49.0042 | 0000.0000.0001 | level-2   |
| p                |      10.0.42.2 |      49.0042 | 0000.0000.0002 | level-2   |
| pe2              |      10.0.42.3 |      49.0042 | 0000.0000.0003 | level-2   |
+------------------+-----------------+--------------+-----------------+-----------+
```

Recap: MPLS Table on PE1

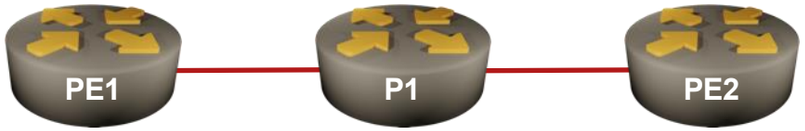
```
pe1# show mpls table
 Inbound Label  Type         Nexthop                    Outbound Label
 -----------------------------------------------------------------------
 15000          SR (IS-IS)   192.168.42.0               implicit-null
 15001          SR (IS-IS)   fe80::a8c1:abff:fe25:ee78  implicit-null
 16002          SR (IS-IS)   192.168.42.0               implicit-null
 16003          SR (IS-IS)   192.168.42.0               16003
 16102          SR (IS-IS)   fe80::a8c1:abff:fe25:ee78  implicit-null
 16103          SR (IS-IS)   fe80::a8c1:abff:fe25:ee78  16103
```

Recap: IPv6 Routing Table on PE1

```
pe1# show ipv6 route
Codes: K - kernel route, C - connected, L - local, S - static,
       R - RIPng, O - OSPFv3, I - IS-IS, B - BGP, N - NHRP,
       T - Table, v - VNC, V - VNC-Direct, A - Babel, F - PBR,
       f - OpenFabric, t - Table-Direct,
       > - selected route, * - FIB route, q - queued, r - rejected, b - backup
       t - trapped, o - offload failure

IPv6 unicast VRF default:
C>* 2001:db8:0:1::/64 is directly connected, lo, weight 1, 00:05:35
L>* 2001:db8:0:1::1/128 is directly connected, lo, weight 1, 00:05:35
I>* 2001:db8:0:2::/64 [115/20] via fe80::a8c1:abff:fe25:ee78, eth1, label implicit-null, weight 1, 00:05:31
I>* 2001:db8:0:3::/64 [115/30] via fe80::a8c1:abff:fe25:ee78, eth1, label 16103, weight 1, 00:05:31
C>* 2001:db8:cafe::/64 is directly connected, eth1, weight 1, 00:05:34
L>* 2001:db8:cafe::2/128 is directly connected, eth1, weight 1, 00:05:34
I>* 2001:db8:cafe:1::/64 [115/20] via fe80::a8c1:abff:fe25:ee78, eth1, weight 1, 00:05:31
C>* fe80::/64 is directly connected, eth1, weight 1, 00:05:33
```

SR-MPLS with Unnumbered Interfaces

Does It Work with Unnumbered Interfaces?



Built-in address pools

- Loopback, point-to-point, LAN
- IPv4 and IPv6
- Pool prefix and allocation prefix (/30 for IPv4 P2P)

Unnumbered interfaces

- Set **ipv4** or **ipv6** to True
- IPv4 interfaces use loopback IPv4 address (also works in VRFs)
- IPv6 interfaces use LLA

topology.yml

```
provider: clab
defaults.device: eos

addressing.p2p.ipv4: True

module: [ isis, sr ]

nodes: [ pe1, p, pe2 ]

links: [ pe1-p, p-pe2 ]

tools:
  edgeshark:
```

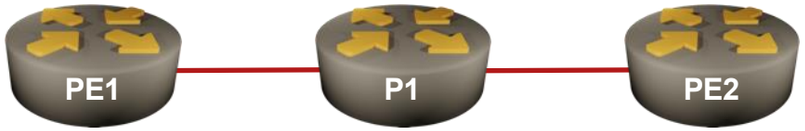
Demo time: SR-MPLS with unnumbered interfaces and IS-IS

Recap: Inspecting SR-MPLS Adjacency Segments

```
pe1#show isis segment-routing adjacency-segments

System ID: pe1                  Instance: Gandalf
SR supported Data-plane: MPLS                   SR Router ID: 10.0.0.1
Adj-SID allocation mode: SR-adjacencies
Adj-SID allocation pool: Base: 100000     Size: 16384
Adjacency Segment Count: 1
Flag Descriptions: F: IPv6 address family, B: Backup, V: Value
                   L: Local, S: Set

Segment Status codes: L1 - Level-1 adjacency, L2 - Level-2 adjacency, P2P - Point-to-Point adjacency, LAN - Broadcast adjacency
                      ! - SR-unreachable, # - Some IS-IS next-hops are SR-unreachable

Locally Originated Adjacency Segments
   Adj IP Address       Local Intf          SID       SID Source                    Flags         Type     Protection
------------------- ---------------- ----------- ---------------- ------------------------- ----------- -----------
        10.0.0.2             Et1       100000          Dynamic       F:0 B:0 V:1 L:1 S:0       P2P L2    unprotected

pe1#
```

SR-MPLS with OSPFv2

Does It Work with OSPFv2?



- Segment routing module (**sr**) works with IS-IS (**isis**) or OSPFv2/OSPFv3 (**ospf**) modules
- IS-IS is the default SR-MPLS protocol
- You have to change **sr.protocol** to use OSPFv2
- *netlab* automatically checks device support and rejects to start a lab with unsupported devices

(Mostly) useless fact

- You can change **sr.protocol** globally or per-node

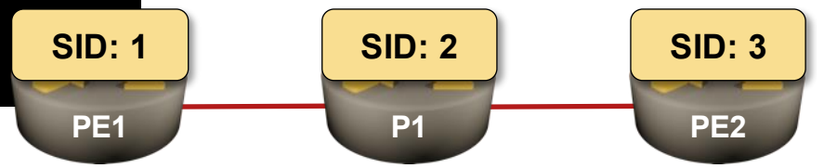
topology.yml

```
provider: clab
defaults.device: eos

module: [ ospf, sr ]
sr.protocol: ospfv2

nodes: [ pe1, p, pe2 ]

links: [ pe1-p, p-pe2 ]
```


Demo time: SR-MPLS with OSPFv2

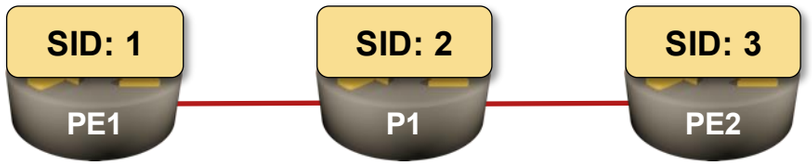
Recap: OSPFv2 Segment Routing Prefix Segments

```
pe1#show ip ospf segment-routing prefix-segments
OSPF Instance ID: 1
SR supported Data-plane: MPLS                    SR Router ID: 10.0.0.1

Node: 3      Proxy-Node: 0      Prefix: 0       Total Segments: 3

Flag Descriptions: NP: No-PHP, M: Mapping Server, E: Explicit-NULL,
                   V: Value, L: Local
Segment status codes: * - Self originated Prefix
  Prefix      SID Type Flags                  Router ID Protection

* 10.0.0.1/32   1 Node NP:0 M:0 E:0 V:0 L:0 10.0.0.1  unprotected
  10.0.0.2/32   2 Node NP:0 M:0 E:0 V:0 L:0 10.0.0.2  unprotected
  10.0.0.3/32   3 Node NP:0 M:0 E:0 V:0 L:0 10.0.0.3  unprotected


pe1#
```



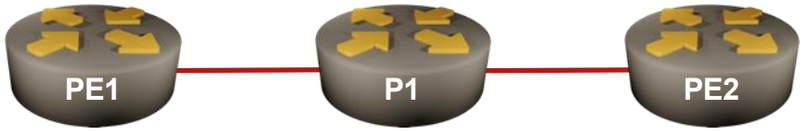
Recap: OSPFv2 Segment Routing Label Bindings

```
pe1#show ip ospf segment-routing bindings
10.0.0.1/32
   Local binding:  Label: imp-null
   Remote binding: Peer ID: 10.0.0.2, Label: 900001
10.0.0.2/32
   Local binding:  Label: 900002
   Remote binding: Peer ID: 10.0.0.2, Label: imp-null
10.0.0.3/32
   Local binding:  Label: 900003
   Remote binding: Peer ID: 10.0.0.2, Label: 900003
pe1#
```



Multivendor SR-MPLS

Multivendor SR-MPLS



You can specify the device type

- Globally (**defaults.device)**
- For each node (using the node **device** parameter)
- For a group of nodes

Off-topic: Selecting the virtualization provider

- System default, lab topology, or individual node
- netlab supports a mix of virtual machines and containers
- You can also use virtual machines in *vrnetlab* containers

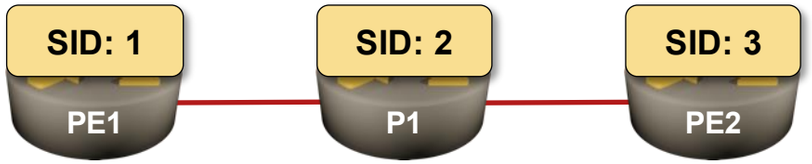
topology.yml

```
provider: clab
module: [ isis, sr ]

nodes:
  pe1:
    device: eos
  p:
    device: frr
  pe2:
    device: crpd

links: [ pe1-p, p-pe2 ]
```

Demo time: Multi-Vendor SR-MPLS with IS-IS

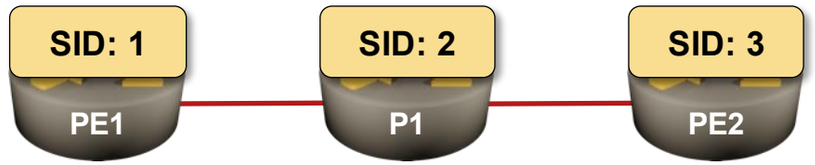
Recap: Vendors Use Different Segment Routing Global Blocks

```
pe1#show isis segment-routing global-blocks

c - conflicting SR capability TLV, processed first advertisement
System ID: pe1                   Instance: Gandalf
SR supported Data-plane: MPLS                    SR Router ID: 10.0.0.1
SR Global Block( SRGB ) Size: 65536

Number of IS-IS segment routing capable nodes excluding self: 2

          SystemId          Base     Size
------- --------------- ------------ -----
   *              pe1        900000    65536
                   p         16000     8000
                 pe2            16     4096
```



Recap: LFIBs Are Built from SRGB + Index Values

```
pe1#show mpls route 900003 | begin 900003
 900003  A[1]
                via M, 10.1.0.1, swap 16003
                    EgressACL: apply
                    directly connected, Ethernet1
                    aa:c1:ab:30:53:77, vlan 1006
pe1#show isis segment-routing tunnel
   Index       Endpoint           Next Hop/Tunnel Index       Interface    Labels
---------- ---------------- ---------------------------- --------------- ---------
   1           10.0.0.2/32        10.1.0.1                    Ethernet1    [ 3 ]
   2           10.0.0.3/32        10.1.0.1                    Ethernet1    [ 16003 ]
```




Want More Fun?
Let's Go…

BGP-Free Core with SR-MPLS

BGP-Free Core

topology.yml

```
provider: clab
defaults.device: eos
bgp.as: 65000

groups:
  _auto_create: True
  core:
    members: [ p1 ]
    module: [ isis, sr ]
  edge:
    members: [ pe1, pe2 ]
    module: [ isis, bgp, sr ]
  hosts:
    members: [ ha, hb ]
    device: linux
```

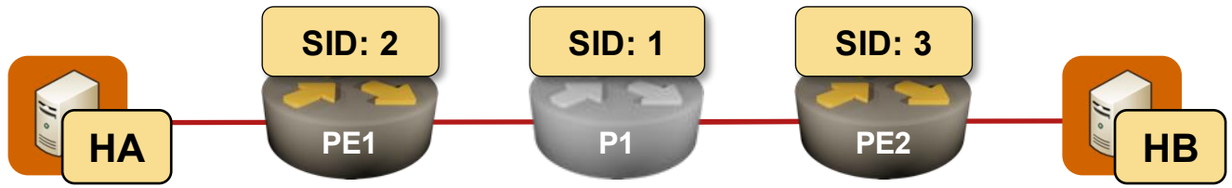

```
links:
- group: core
  members: [ pe1-p, p-pe2 ]
- group: edge
  isis: False
  bgp.advertise: True
  members: [ ha-pe1, hb-pe2 ]
```

Download from https://github.com/ipspace/SR-workshop/tree/main/2-fun/1-bgp-free


Demo time: BGP-Free Core

Recap: SR-MPLS Labels Are Attached to BGP Routes

```
pe1#show ip route 172.16.1.0 | begin 172.16
 B I      172.16.1.0/24 [200/0]
           via 10.0.0.3/32, IS-IS SR tunnel index 1
              via 10.1.0.1, Ethernet1, label 900003
```



© ipSpace.net 2026

L3VPN with SR-MPLS

L3VPN (RFC 4364) with SR-MPLS Core

topology.yml

```
bgp.as: 65000
mpls.vpn: True
mpls.ldp: False

groups:
  _auto_create: True
  core:
    members: [ p1 ]
    module: [ isis, sr ]
  edge:
    members: [ pe1, pe2 ]
    module: [ isis, bgp, sr, mpls, vrf ]
  hosts:
    members: [ ha, hb ]
    device: linux
```

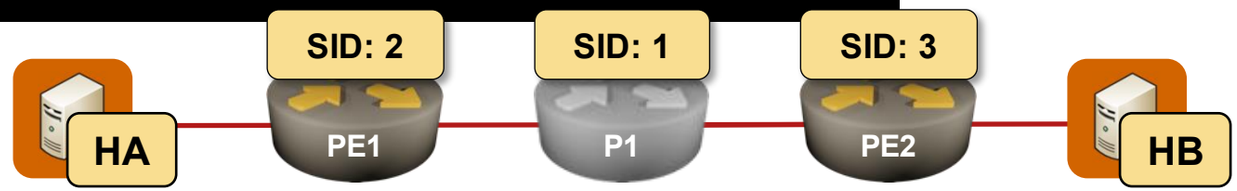

```
vrfs:
  tenant:
    links: [ ha-pe1, hb-pe2 ]

links:
- group: core
  members: [ pe1-p1, p1-pe2 ]
```

Download from https://github.com/ipspace/SR-workshop/tree/main/2-fun/2-mpls-vpn

Demo time: MPLS/VPN with SR-MPLS Core

Recap: SR-MPLS Labels Are Used as First Labels in VRF Routes

```
pe1#show bgp vpn-ipv4 172.16.1.0/24
BGP routing table information for VRF default
Router identifier 10.0.0.2, local AS number 65000
BGP routing table entry for IPv4 prefix 172.16.1.0/24, Route Distinguisher: 65000:1
 Paths: 1 available
  Local
    10.0.0.3 from 10.0.0.3 (10.0.0.3)
      Origin IGP, metric -, localpref 100, weight 0, tag 0, valid, internal, best
      Extended Community: Route-Target-AS:65000:1
      Remote MPLS label: 116384
pe1#show ip route vrf tenant 172.16.1.0/24 | begin 172
 B I      172.16.1.0/24 [200/0]
          via 10.0.0.3/32, IS-IS SR tunnel index 1, label 116384
             via 10.1.0.1, Ethernet1, label 900003
```



EVPN with SR-MPLS

EVPN with SR-MPLS Core

topology.yml

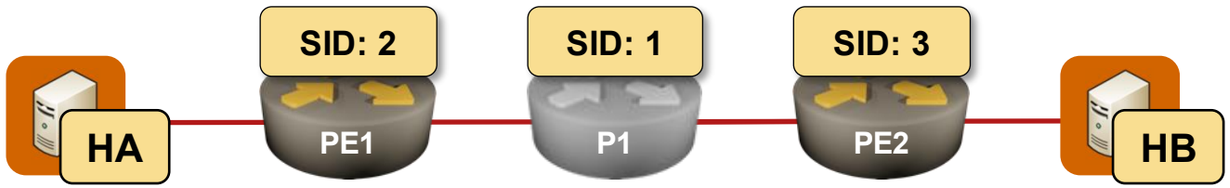
```
bgp.as: 65000
evpn.transport: sr

groups:
  _auto_create: True
  core:
    members: [ p1 ]
    module: [ isis, sr ]
  edge:
    members: [ pe1, pe2 ]
    module: [ isis, bgp, sr, vlan, evpn ]
  hosts:
    members: [ ha, hb ]
    device: linux
```


```
vlans:
  tenant:
    mode: bridge
    links: [ ha-pe1, hb-pe2 ]

evpn.vlans: [ tenant ]

links:
- group: core
  members: [ pe1-p1, p1-pe2 ]
```


Demo time: EVPN with SR-MPLS core

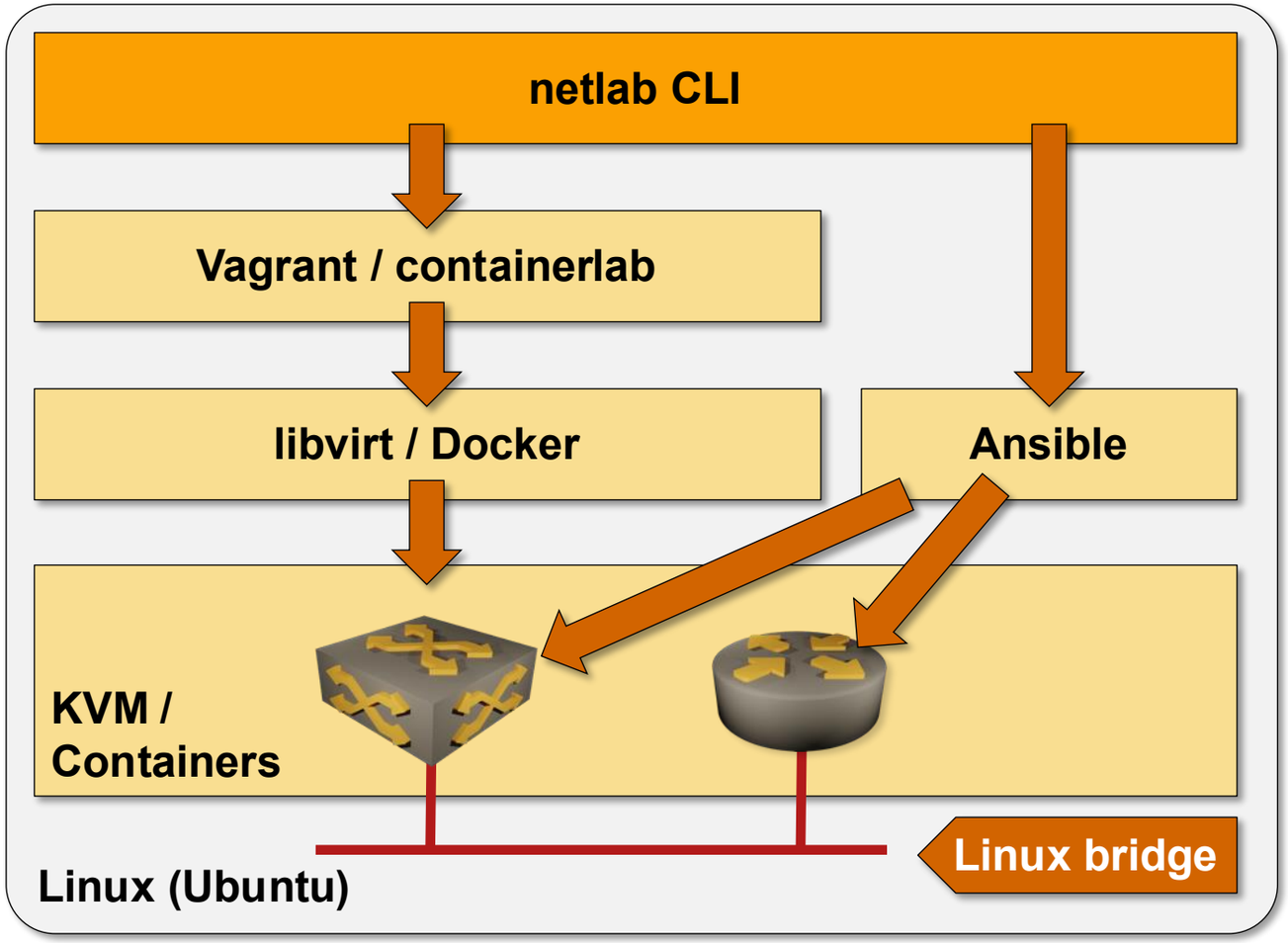
Recap: MAC-VRF RIB Uses SR-MPLS Labels

```
pe1#show bgp evpn route-type mac-ip aac1.abf9.7f8c detail
BGP routing table information for VRF default
Router identifier 10.0.0.2, local AS number 65000
BGP routing table entry for mac-ip aac1.abf9.7f8c, Route Distinguisher: 10.0.0.3:1000
 Paths: 1 available
  Local
    10.0.0.3 from 10.0.0.3 (10.0.0.3)
      Origin IGP, metric -, localpref 100, weight 0, tag 0, valid, internal, best
      Extended Community: Route-Target-AS:65000:1000 TunnelEncap:tunnelTypeMpls
      MPLS label: 1047390 ESI: 0000:0000:0000:0000:0000
pe1#show l2Rib input bgp detail
aac1.abf9.7f8c, VLAN 1000, seq 1, pref 16, EVPN dynamic remote
   Label entry 1: 1047390
      Tunnel IS-IS SR IPv4 (1), TEP 10.0.0.3/32
```



Deployment Scenarios

Recommended: Ubuntu, KVM, libvirt, Docker



Prerequisite software

- Python3
- Ansible (to configure the devices)

Devices as virtual machines

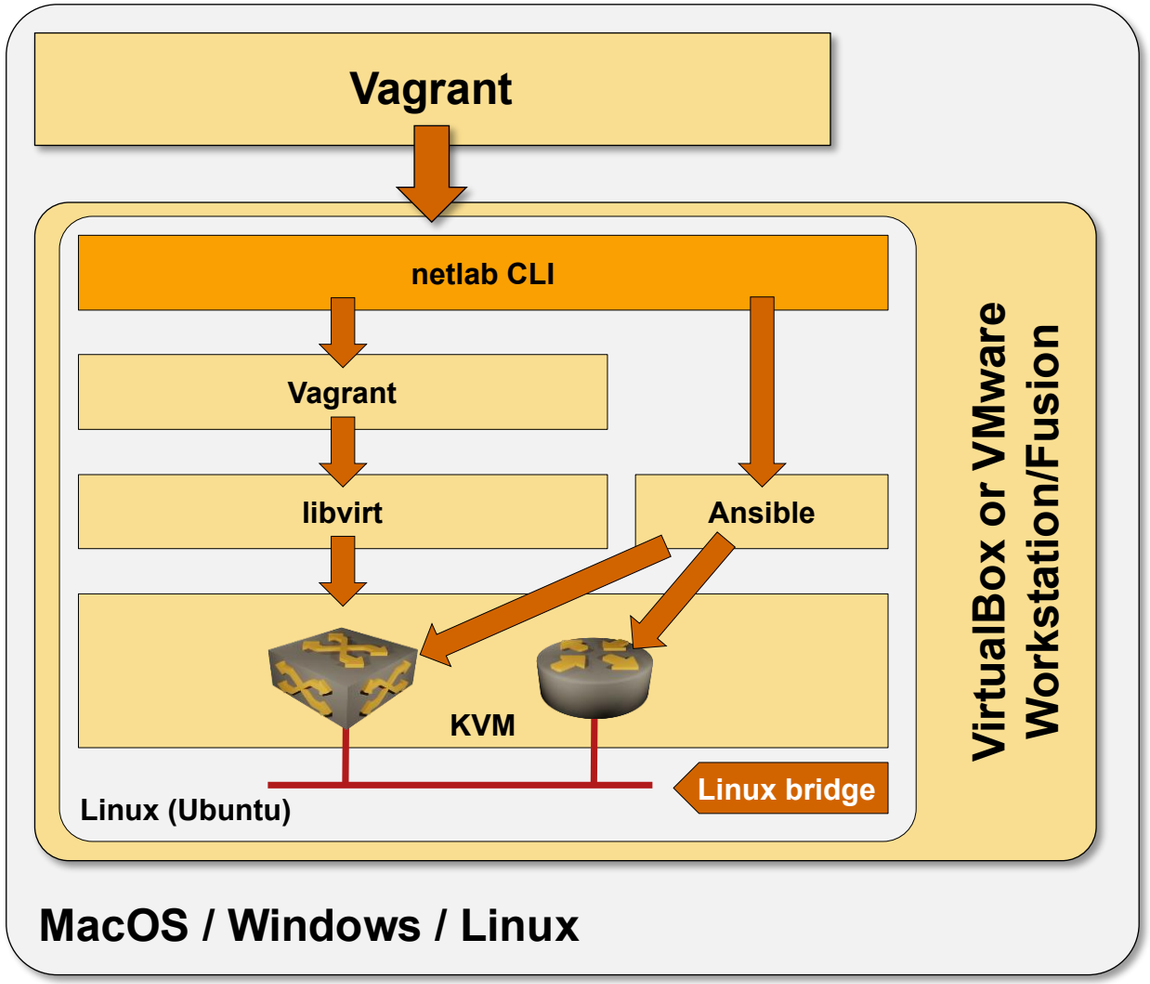
- KVM
- libvirt
- Vagrant with vagrant-libvirt plugin

Containers (including VM-in-container)

- Docker
- Containerlab

Use standalone server, a local VM, a VM in the cloud, or GitHub Codespaces

Use Existing x86 Device: Ubuntu VM



Requirements

- You can run containers with any VM virtualization product
- Nested virtualization is required to run network device VMs

Virtualization solution with nested virtualization

- Hyper-V (WSL)
- KVM
- VirtualBox
- VMware Workstation/Fusion

Optional

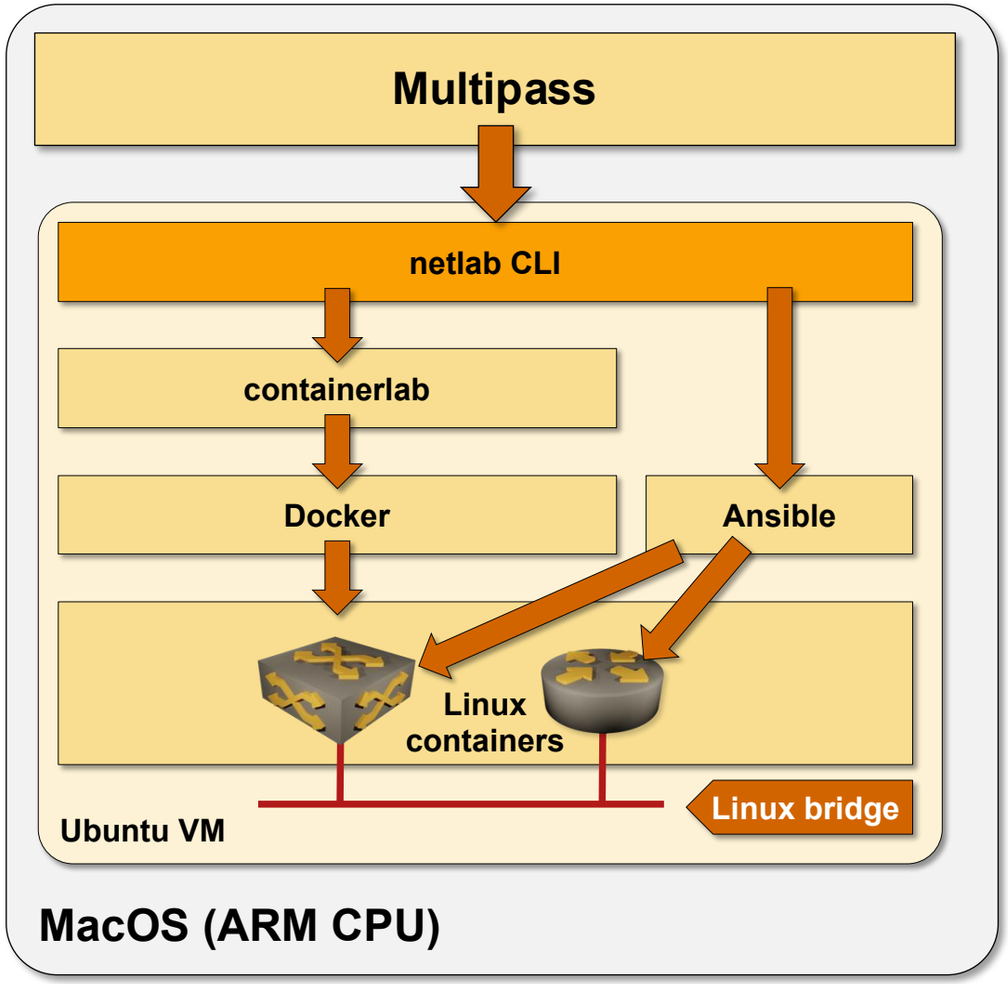
- Start the VM with Vagrant (simplifies the operations)

Alternative

- Cloud deployment

More @ https://netlab.tools/install/ubuntu-vm/

Ubuntu VM on Apple Silicon



- **multipass** starts an Ubuntu VM on an ARM CPU
- Nested virtualization is not supported ➔ containers only
- Container images must be built for the ARM CPU ➔ Arista EOS, FRRouting, and SR Linux

Interesting use cases

- Run BGP, VXLAN, or EVPN labs on your Apple laptop
- FRRouting and Arista configuration syntax are pretty close to the *industry standard CLI*

More @ https://blog.ipspace.net/2024/03/netlab-bgp-apple-silicon/

Some Assembly Required (Thank You, Vendors)

Automatically downloadable images and containers

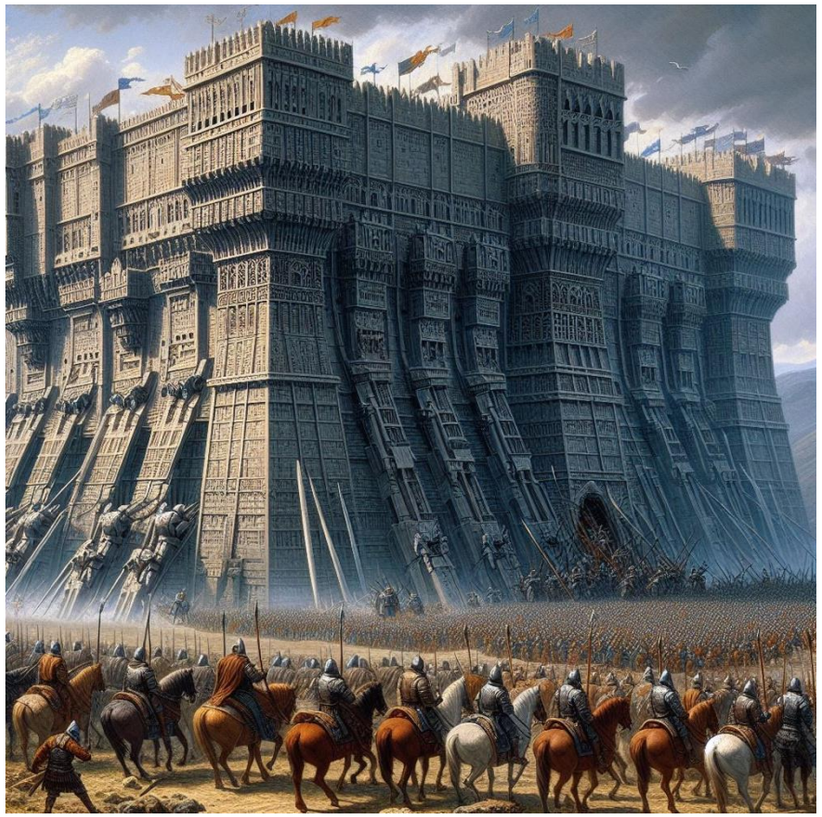
- FRRouting
- Linux, OpenBSD
- Nokia SR Linux
- VyOS

Easy to download (no registration required)

- Juniper (some images), Dell OS10, Mikrotik RouterOS7

Most everything else (from bad to worse)

- Registration (Arista EOS, Aruba CX, Cisco Nexus OS, Cumulus)
- Download tied to a valid support contract (Cisco CSR)
- Begging your SE
- Available only if you know the right dev person



But Wait, That's Not All

After you download a container image (Arista cEOS)

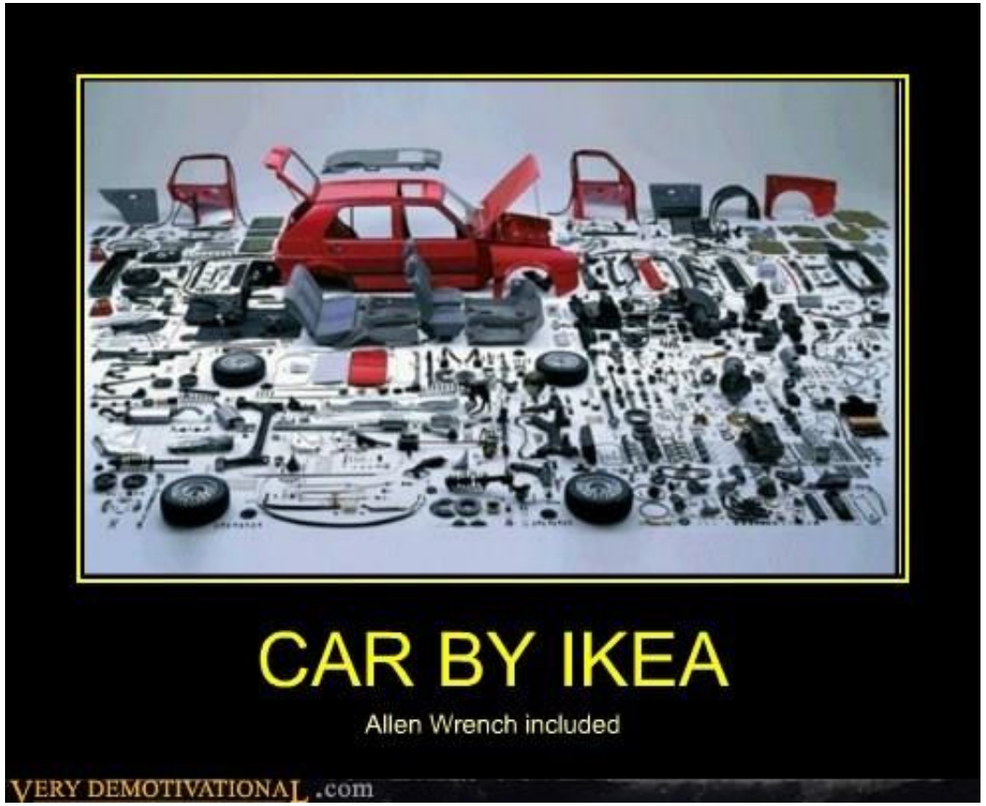
- Unpack and install it (easy)

Virtual machines are a nightmare

- Download a virtual disk
- It boots without a configuration and expects stuff on serial port
- Exception: Junos and ASA can take configuration from a mounted CD-ROM

Building a Vagrant box

- What we need to automate lab startup
- Start the VM, answer a dozen questions
- Copy-paste initial configuration
- Save the configuration, shut down the VM
- Package as a Vagrant box, hope it works




UNICORN
AS A
SERVICE

No Presentation Is Complete Without Mentioning AI/ChatGPT

> netlab is a lab creation tool documented at https://netlab.tools/ with additional examples provided at https://blog.ipspace.net/tag/netlab/ and sample lab topologies at https://github.com/ipspace/netlab-examples.
>
> Please use that documentation to create a netlab topology file for a network with three routers (PE1, P1, PE2) connected PE1-P1 and P1-PE2. The routers should be Arista EOS containers running IS-IS and SR-MPLS.

- Results suck more unless you mention the URLs in the prompt
- The initial result usually doesn't work. Run **netlab** and copy-paste error messages to ChatGPT
- Like any persistent sloppy clueless intern, it will eventually get it right
- It took ChatGPT only two retries (in March 2026) to get a suboptimal (but working) topology

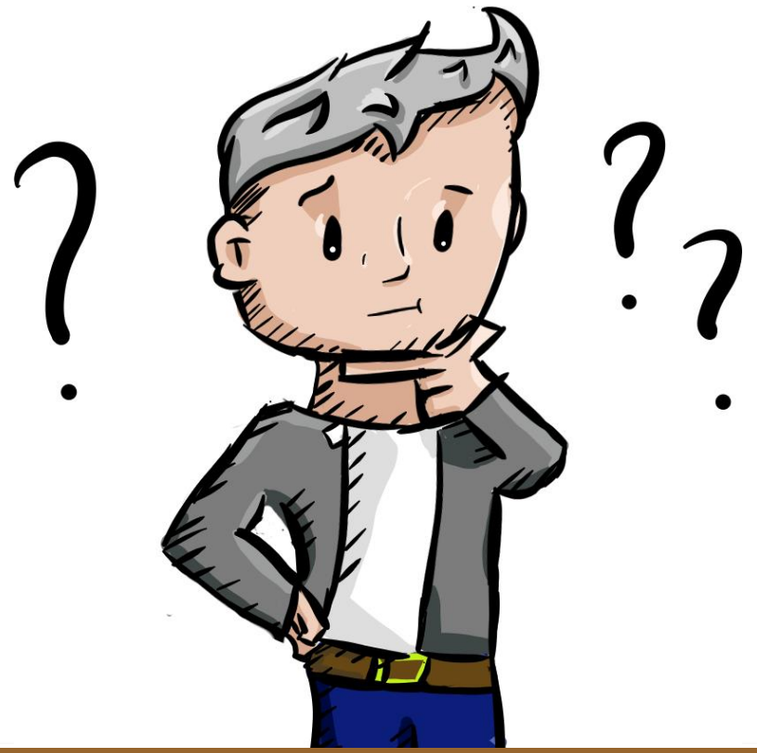
ChatGPT-Generated Topology

```
---
name: sr-mpls-isis-lab

provider: clab

defaults:
  device: eos
  providers:
    clab:
      image: ceos:latest

module: [ isis, sr ]
```

```
nodes:
  PE1:
    isis:
      net: 49.0001.0000.0000.0001.00
  P1:
    isis:
      net: 49.0001.0000.0000.0002.00
  PE2:
    isis:
      net: 49.0001.0000.0000.0003.00

links:
  - PE1-P1
  - P1-PE2
```

Questions?

Documentation:	**netlab.tools**
Blog posts:	blog.ipspace.net/tag/netlab.html
Source code:	github.com/ipspace/netlab
Examples:	github.com/ipspace/netlab-examples
Sample project:	bgplabs.net (BGP labs) isis.bgplabs.net (IS-IS labs) evpn.bgplabs.net (EVPN/VXLAN labs)



To reach me

Web:	ipSpace.net
Email:	ip@ipSpace.net

Use GitHub issues / discussions to ask netlab questions and report challenges