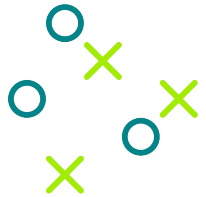



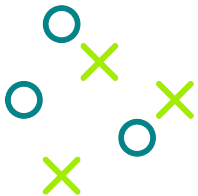
codilime®
CREATING VALUE

The modern, interoperable DC

Part 2: EVPN as a universal solution for VM, container and BMS networking

Adam Kułagowski
Jerzy Kaczmarski

team@codilime.com




Introduction

10
Years in business

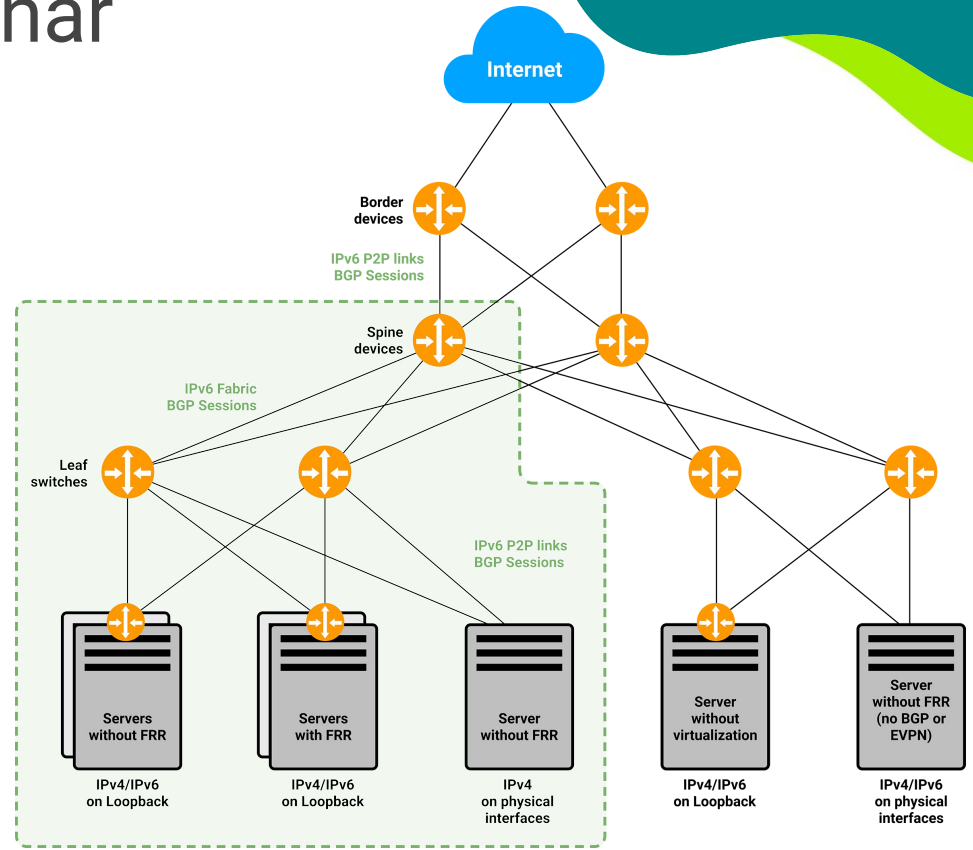
3
Offices

200+
Network, software
& DevOps engineers

6
Our clients'
Time zones

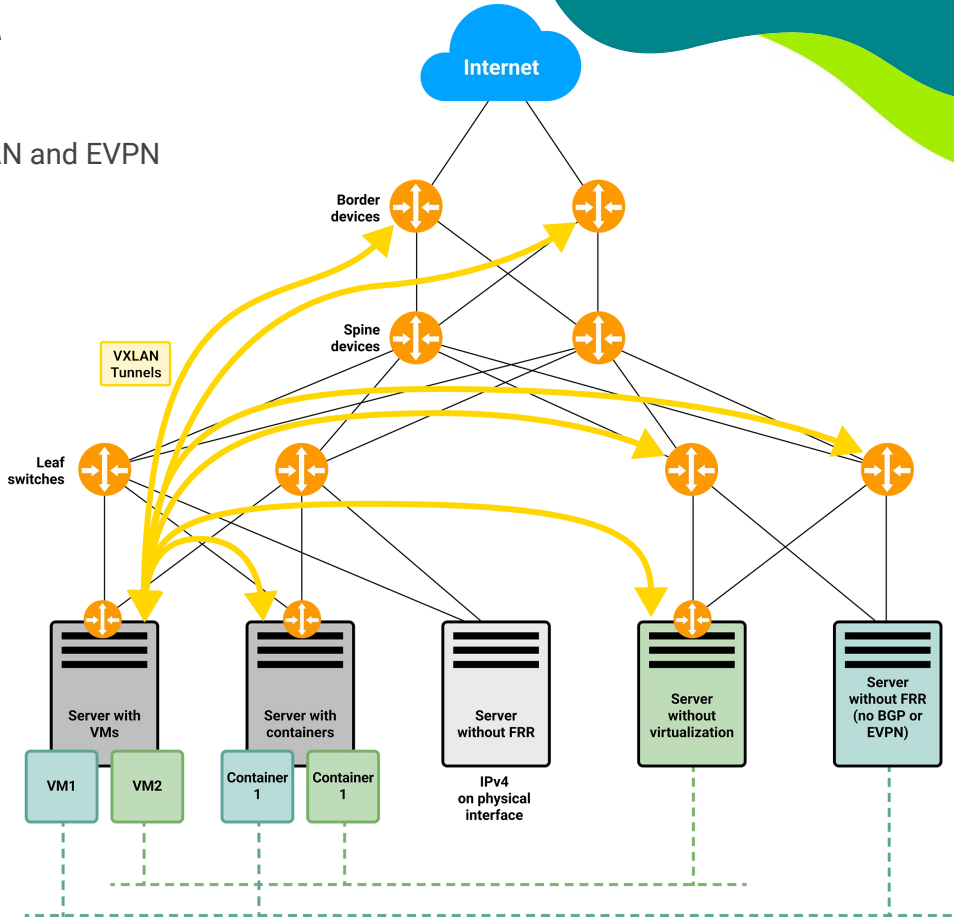
Recap of previous webinar

- Scalable and easily automated networking architecture for DC environments
- Using IPv6 to simplify deployment and address management
- Using BGP for scalability, flexibility, load balancing and fast failover
- Automatically installing and configuring FRR



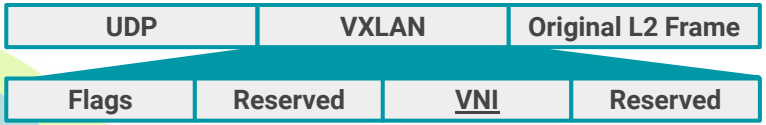
Today’s webinar agenda

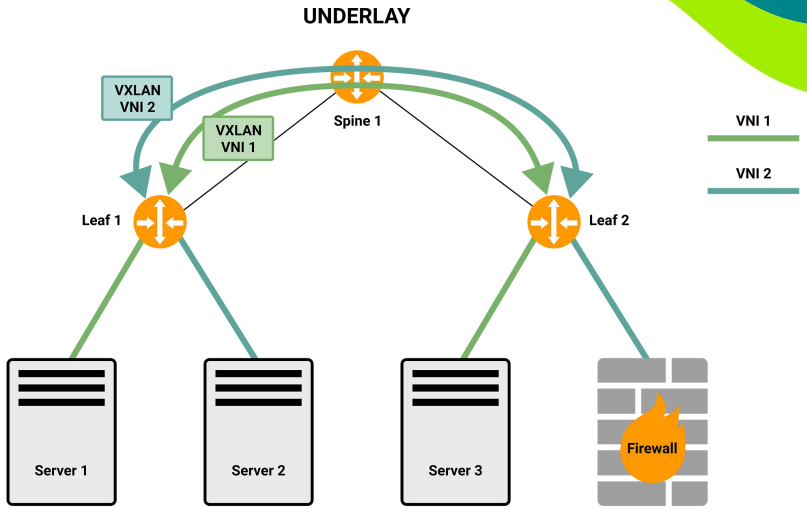
- Providing Layer 2 connectivity over IPv6 fabric with VXLAN and EVPN
- Multi-tenancy using virtualization, VRFs and tunnels
- Interconnecting heterogeneous resources at Layer 2:
  - legacy server without FRR or virtualization
  - server with FRR installed
  - containers (i.e. kubernetes)
  - virtual machines
- Demo of working solution
- Q&A session

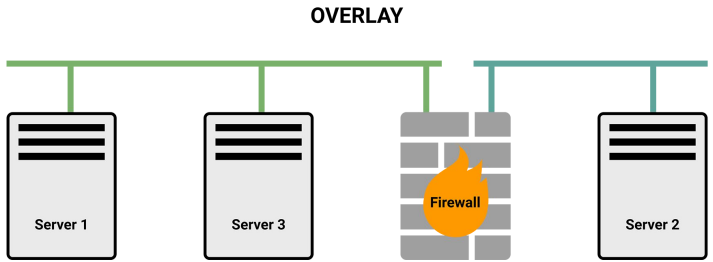


VXLAN tunnels

- Simple protocol, can be supported in software (i.e. Linux kernel) and in hardware (i.e. DC switches)
- Allows tunneling of Ethernet frames over Layer 3 connections
- A mesh of VXLAN tunnels creates an “overlay” network
- Overlay networks separated based on VNI (similar to VLAN in standard Ethernet)
- Tunnel endpoints can be learned/advertised using control plane mechanisms, i.e.:
  - Manual configuration + MAC address learning
  - Multicast routing protocol + MAC address learning
  - SDN controller
  - Ethernet VPN (EVPN)

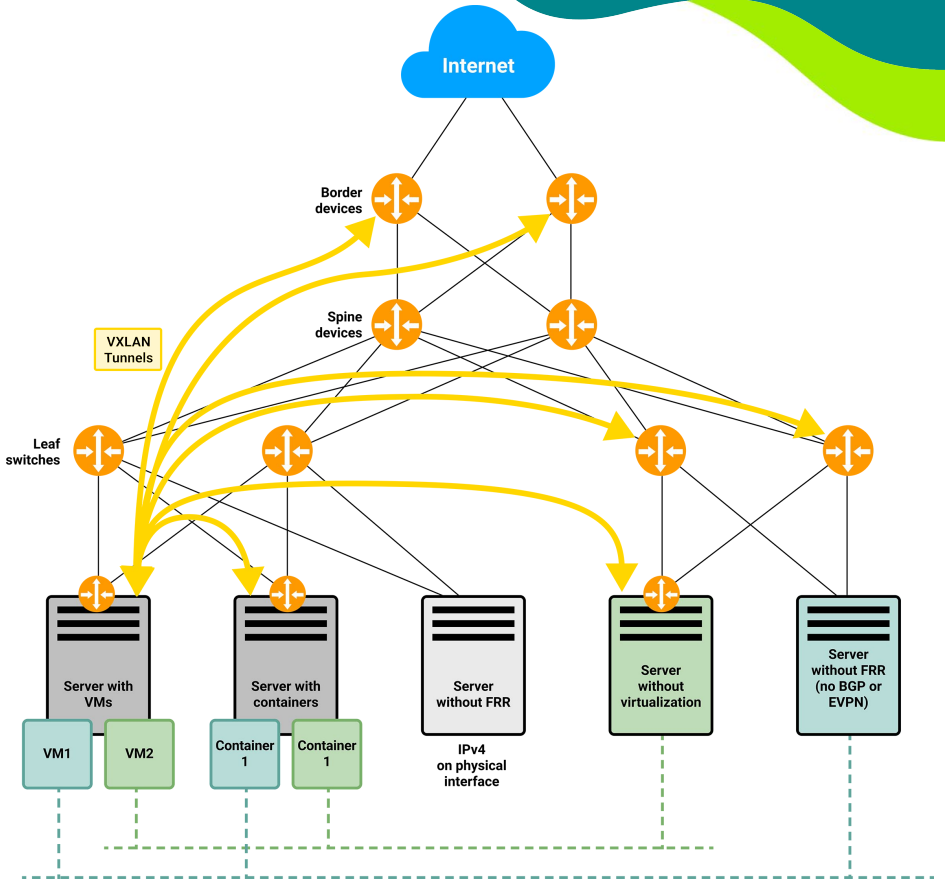






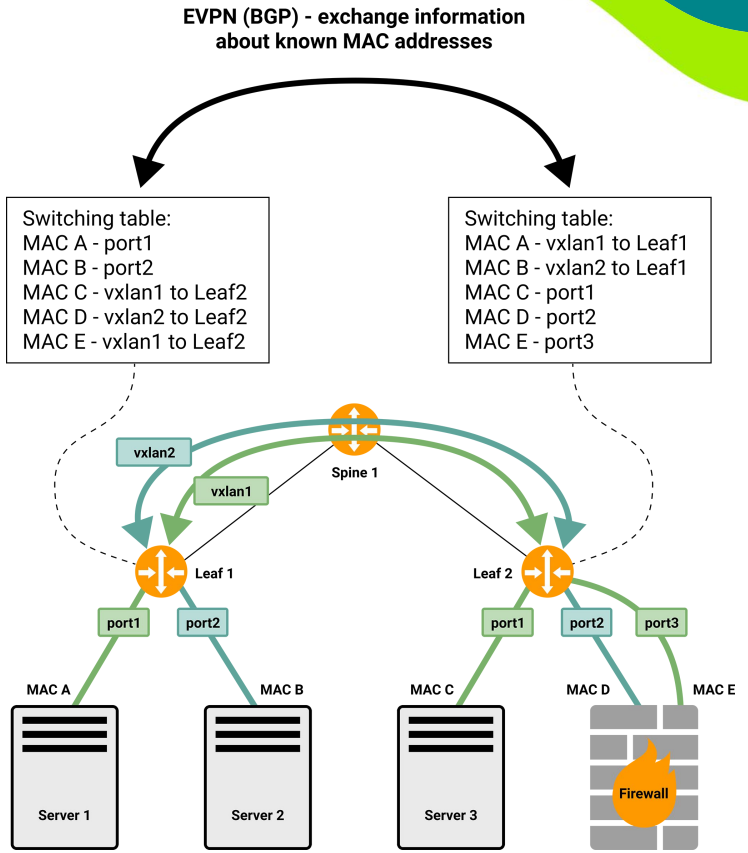
Ethernet VPN

- Standardized protocol to advertise MAC/IP reachability and tunneling information
  - Supports several tunneling mechanisms, including MPLS (mainly ISPs) and VXLAN (mainly DCs)
- Provides additional important features for DCs:
  - MAC/IP advertisement
  - Proxy ARP (less broadcasts)
  - MAC address mobility (i.e. helps with VM migration)
  - Gateway discovery (including distributed gateway)
  - Active/Passive and Active/Active multihoming
  - Layer 3 prefix advertisement (similar to L3VPN)
  - Multitenancy
- Can be run on networking devices and on servers
  - FRR supports EVPN



Connecting BMS together

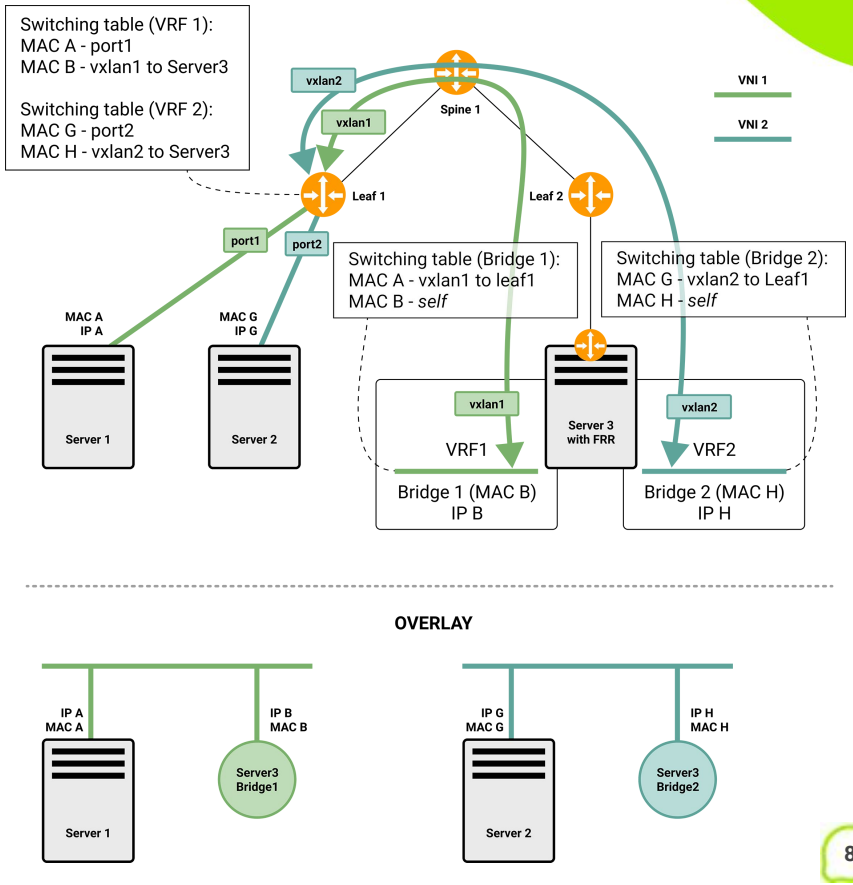
- BMS/Network Appliances in this example do not need any extra configuration
  - They are not aware of VXLAN tunnels or EVPN
- Leaf switches where BMSs are connected require EVPN and VXLAN support
- Ports/VLANs on switches are assigned to VXLAN VNIs
- BGP is enabled and configured for EVPN
- BGP automatically advertises information required for tunneling, i.e.:
  - known MAC/IPs (based on MAC and ARP tables)
  - VXLAN tunnel endpoints and VNIs



Connecting BMS with FRR (1/2)

- FRR supports EVPN including data plane integration (i.e. adding MACs to Linux bridge tables)
- Linux supports multiple bridges on single server (Layer 2 separation/multitenancy)
- Linux supports multiple routing tables or VRFs (Layer 3 separation/multitenancy)
- EVPN advertises MAC/IP information with so-called Route Targets (which allows for separation/multitenancy)
- Example separation:
  - Network 1 - VNI 1, Route Target 65000:1 (VRF 1)
  - Network 2 - VNI 2, Route Target 65000:2 (VRF 2)

Note: in this scenario Leaf2 does not need to support EVPN or VXLAN

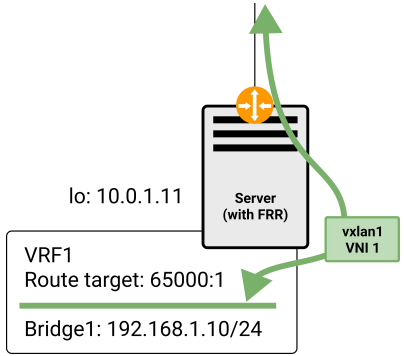


Connecting BMS with FRR (2/2)

- Example of adding a new Linux bridge to EVPN on FRR
  - Prerequisite - FRR installed on server, BGP neighbors already configured

```
# Create new bridge
ip link add name bridge1 type bridge
# Add IP address to new bridge
ip address add 192.168.1.10/24 dev bridge1
# (Optional) Add bridge to separate VRF to provide multitenancy
ip link add vrf1 type vrf table 1
ip link set dev vrf1 up
ip link set dev bridge1 master vrf1
# Create VXLAN interface and attach it to bridge1
ip link add vxlan1 type vxlan id 1 dstport 4789 local 10.0.1.11 proxy
nolearning
ip link set dev vxlan1 up
ip link set dev vxlan1 master bridge1

# Configure FRR to advertise MAC/IPs learned on this bridge
address-family l2vpn evpn
  vni 1
       rd 10.0.1.11:1
       route-target import 65000:1
       route-target export 65000:1
  exit-vni
  advertise-all-vni
```



Connecting VMs to EVPN

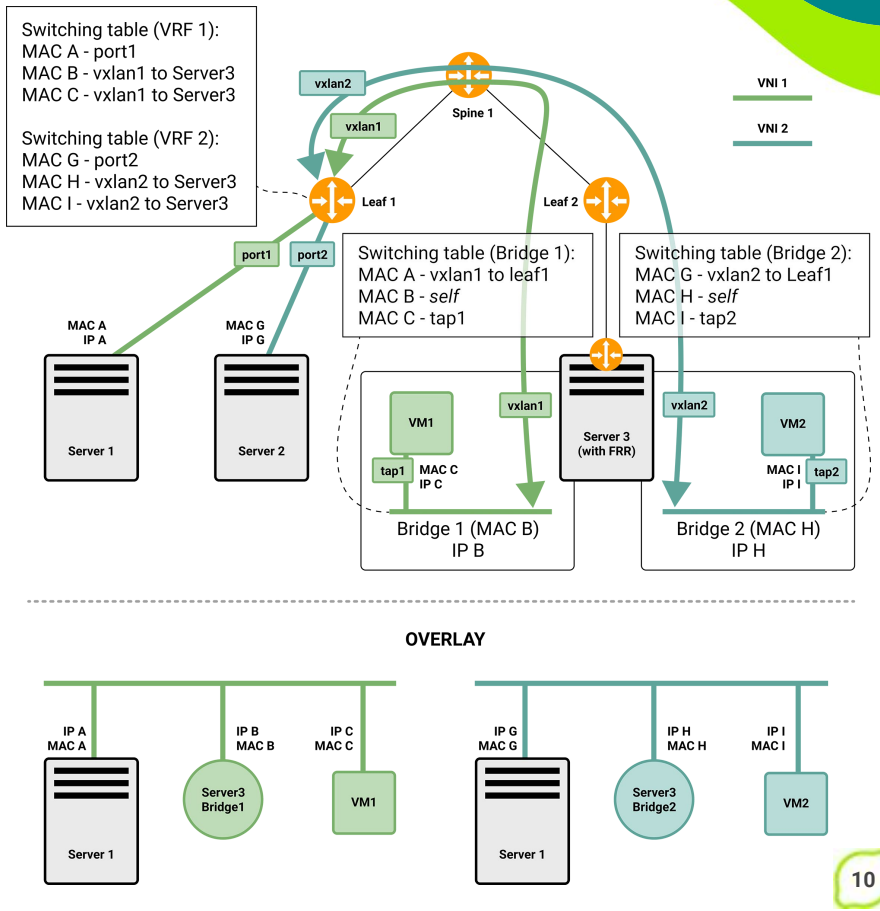
- Very similar to previous example
- One extra step:
  - Connect VMs to bridges that are attached to EVPN

```
ip link set dev tap1 master bridge1
ip link set dev tap2 master bridge2
```

- Supported by libvirt:

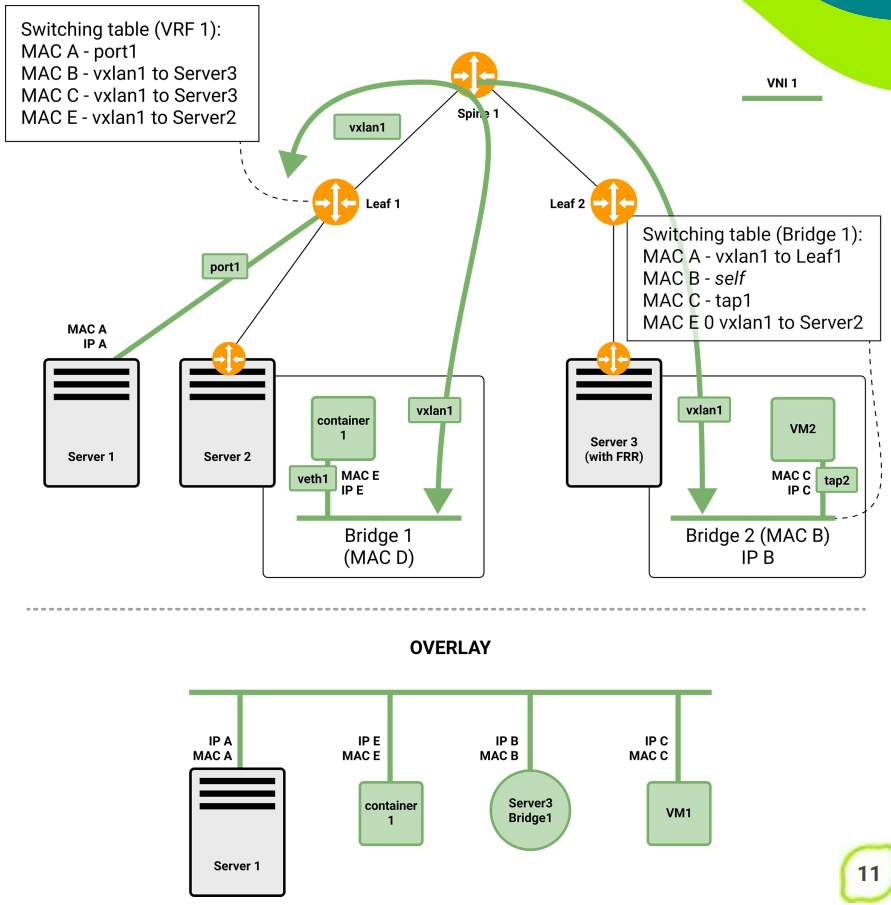
```
virsh # domiflist --domain alpine-blue
 Interface     Type     Source    ModelMAC
-----------------------------------------------------------
 vnet0         network   blue        virtio    52:54:00:03:82:d8

virsh # domiflist --domain alpine-green
 Interface    Type      Source    Model MAC
-----------------------------------------------------------
 vnet1         network   greenvirtio    52:54:00:32:2a:a7
```



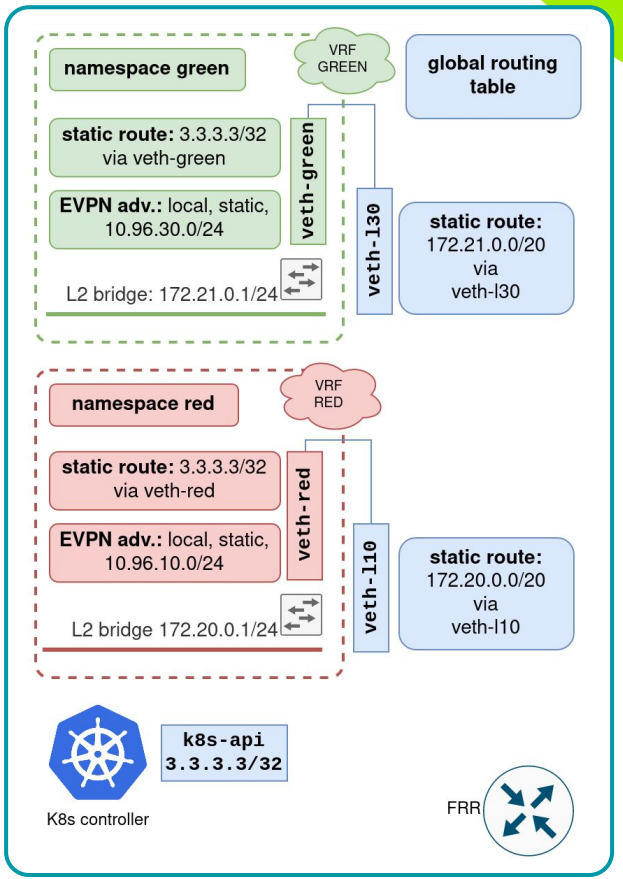
Connecting containers to EVPN (1/2)

- Each container is placed in a dedicated namespace
- Each namespace is a separate L2/L3 FWD table
- To connect namespace between each other we can use veth interfaces
- We can attach the veth interface placed in GRT to the bridge bonded to VRF



Connecting containers to EVPN (2/2)

- Using **kube-namespace-cni,** pods are attached to a specific bridge based on namespace
- Each bridge is attached to a different **Linux VRF**
- Using veth interfaces, **route leaking** between VRF and global routing table is possible, so kubelets can monitor the pods health
- FRR advertises, via EVPN, within each VRF:
  - Connected networks (pods IPs)
  - Static route to local kubelet IP
  - Unique part of ClusterIP network (i.e 10.96.10.0/24 for VRF red) for service exposure outside of K8s

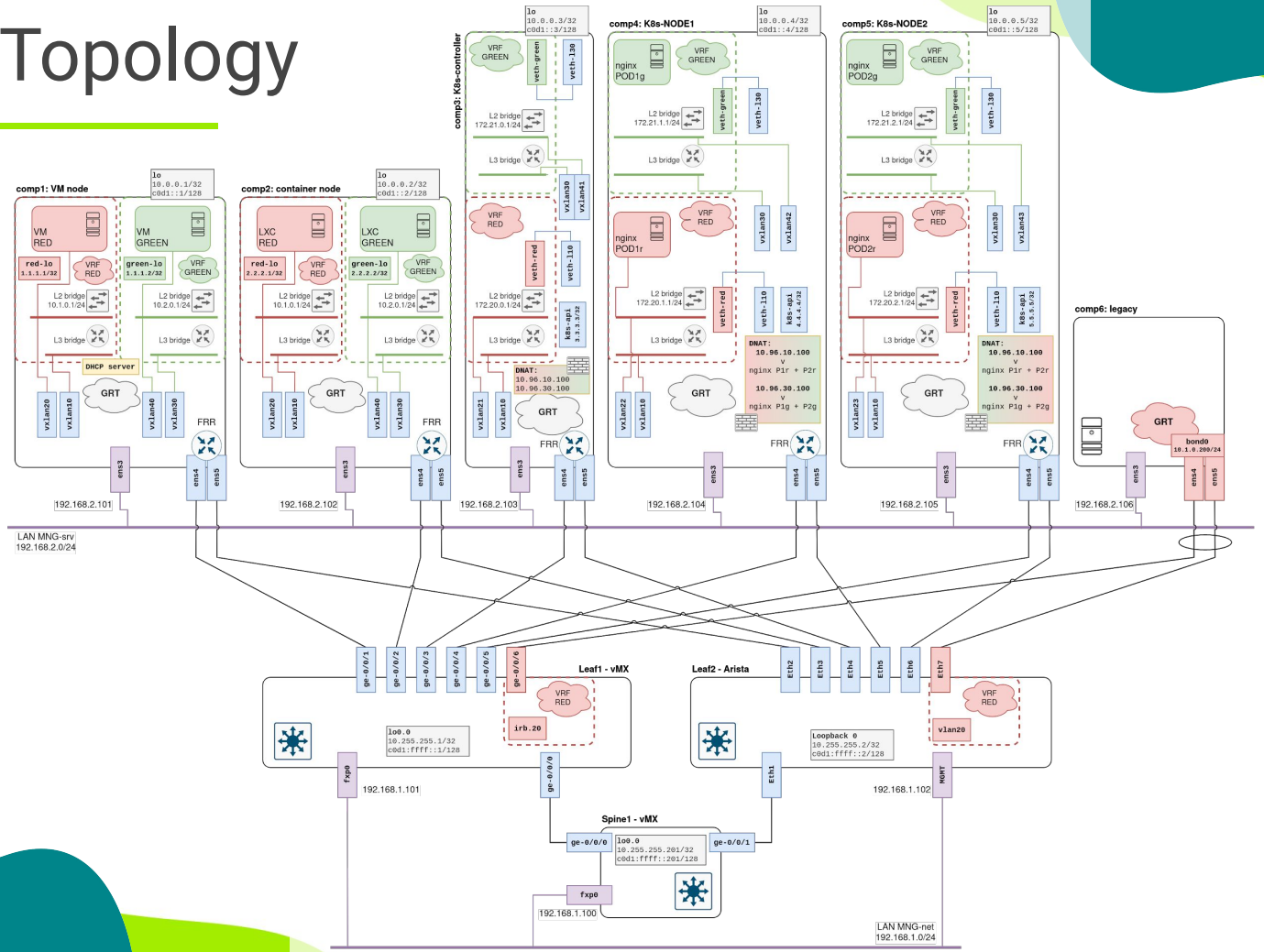


Demo

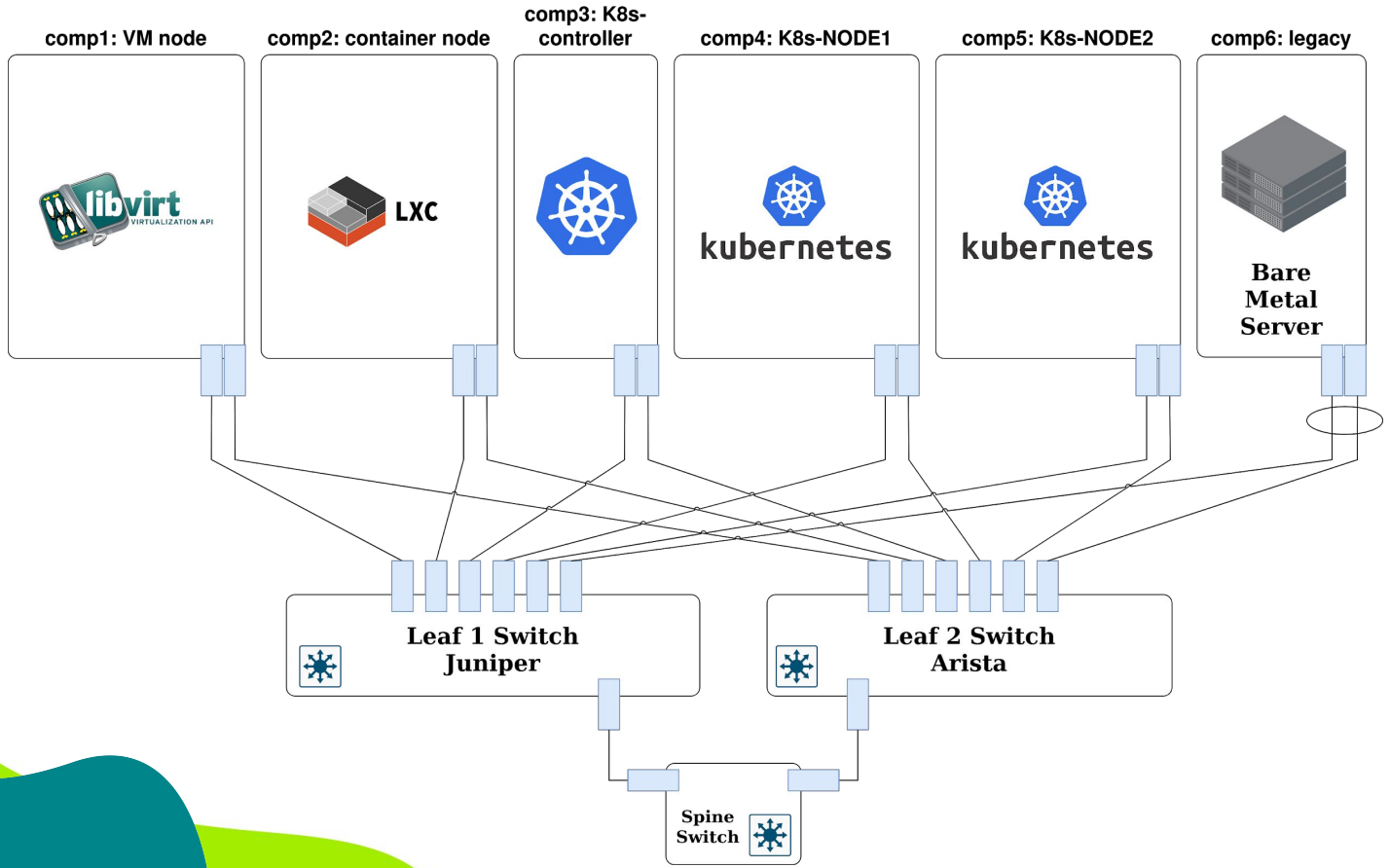
Demo agenda

- Topology presentation
- Creating & starting 2x LXC for GREEN & RED VRFs
- Communication between VM and LXC
- K8s deployment of 2x nginx inside VRF RED
- K8s deployment of 2x nginx inside VRF GREEN
- Service creation w/ ClusterIP for VRF RED
- Service creation w/ ClusterIP for VRF GREEN
- Access from VM RED to nginx service on K8s (w/ load balancing)
- Access from legacy BMS to nginx service on K8s (RED & GREEN)

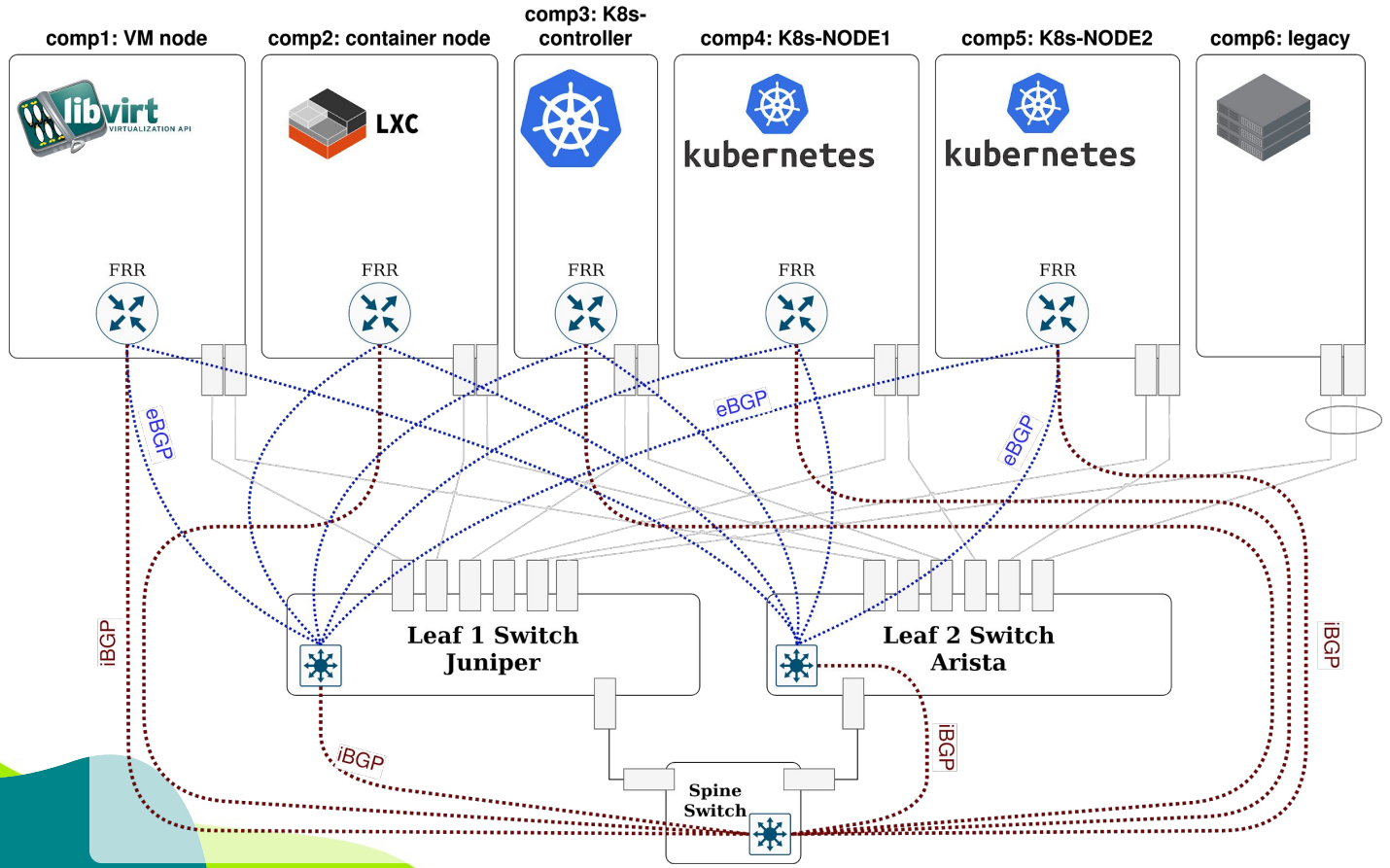
Demo Topology



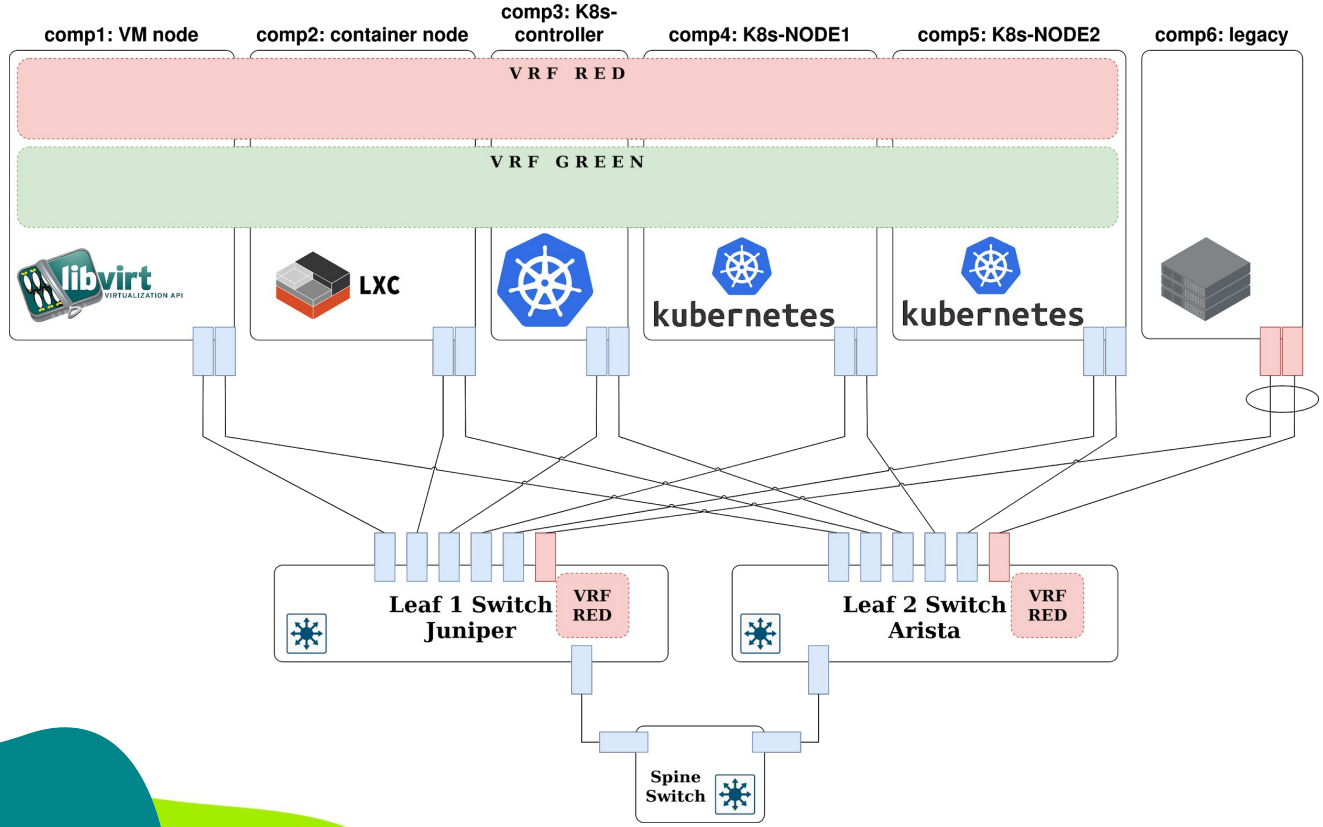
Demo Topology



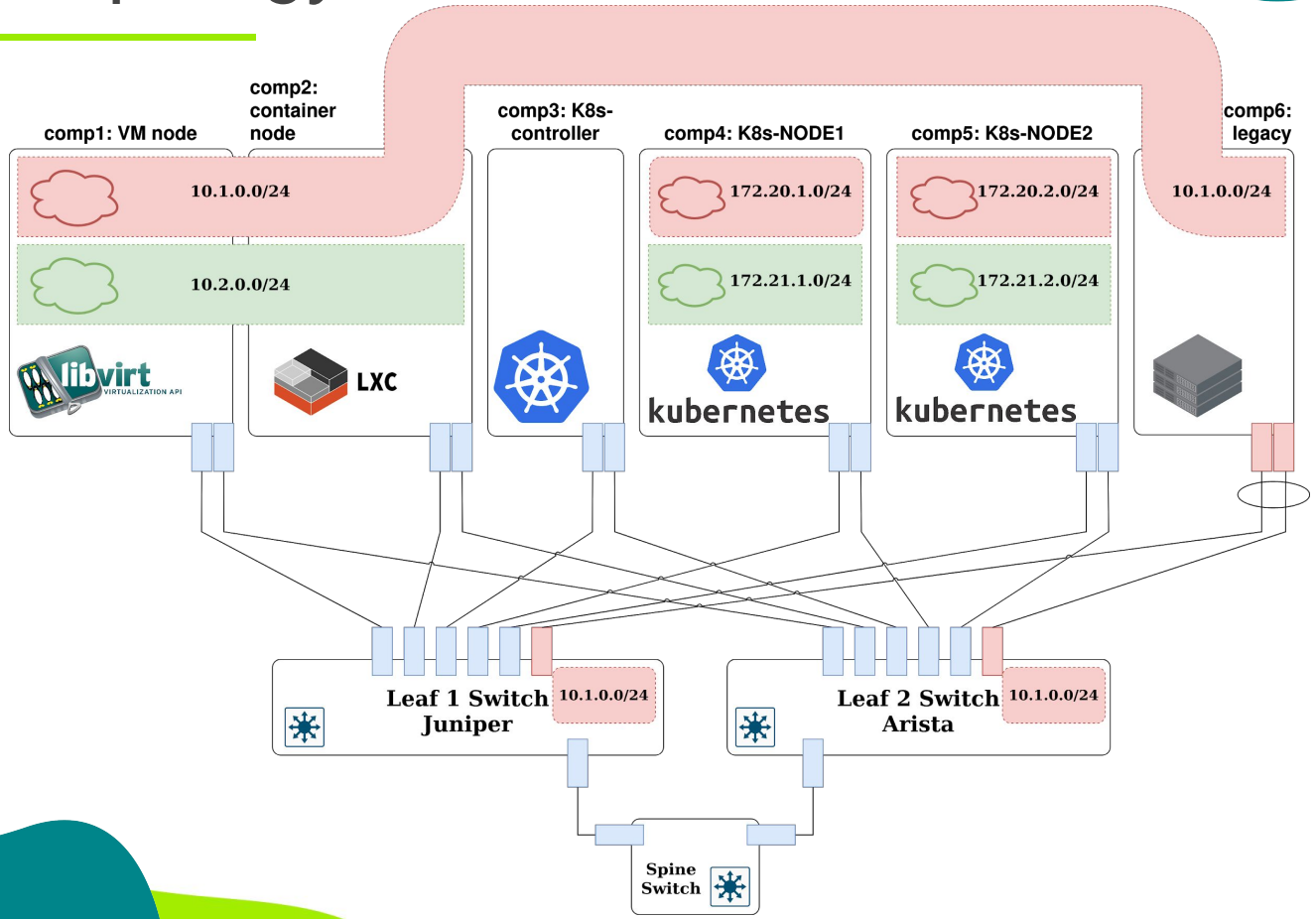
Demo Topology



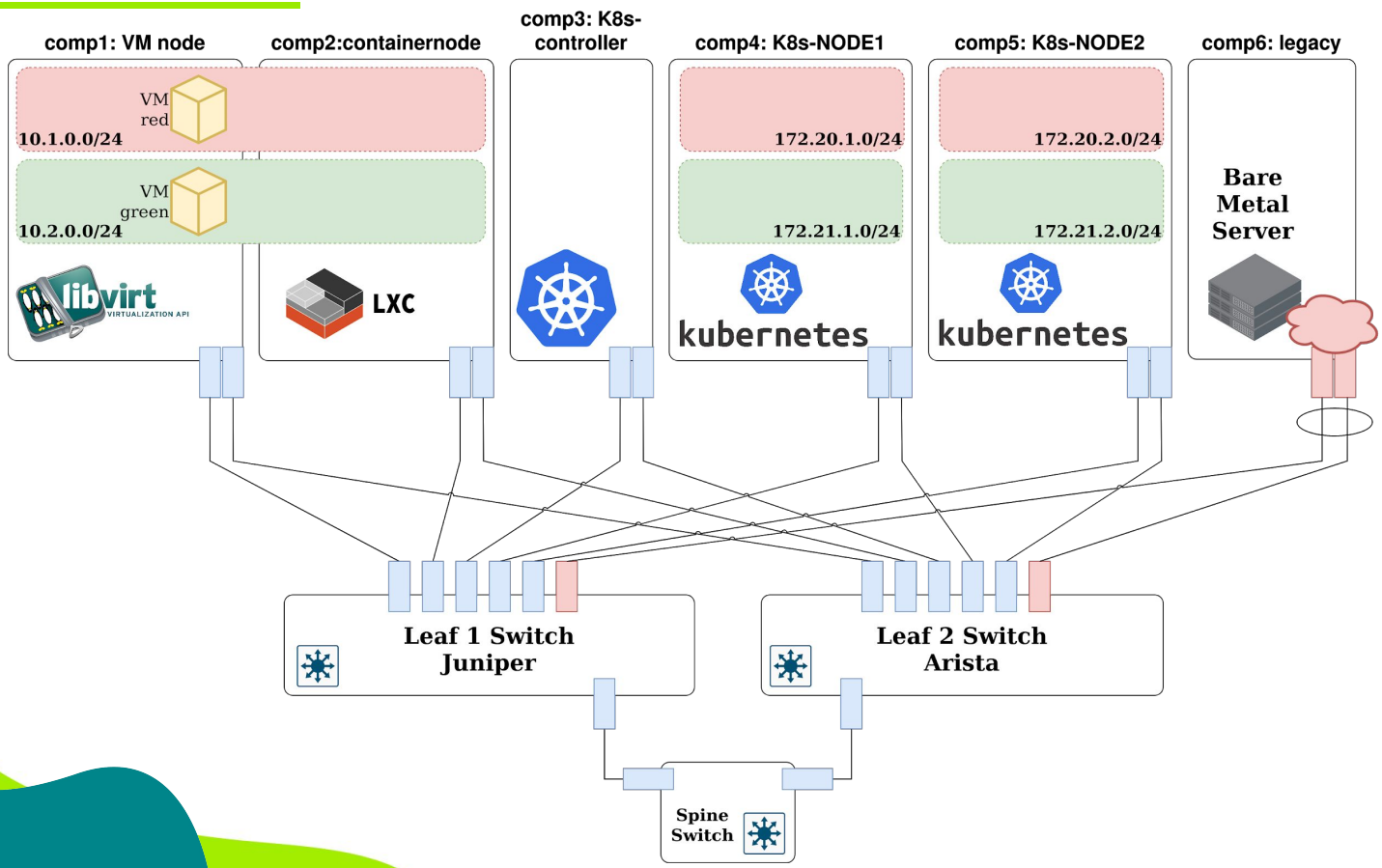
Demo Topology



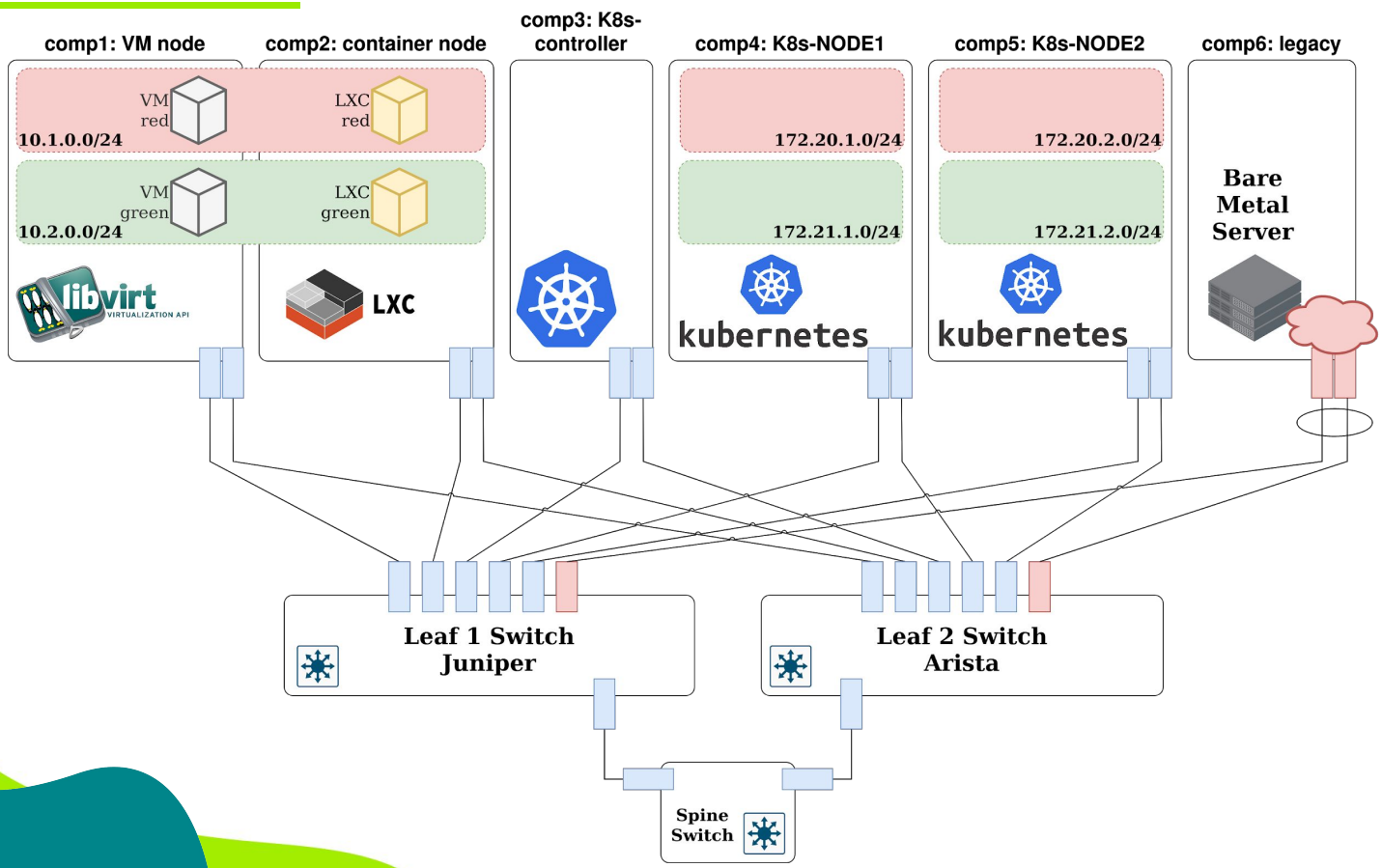
Demo Topology



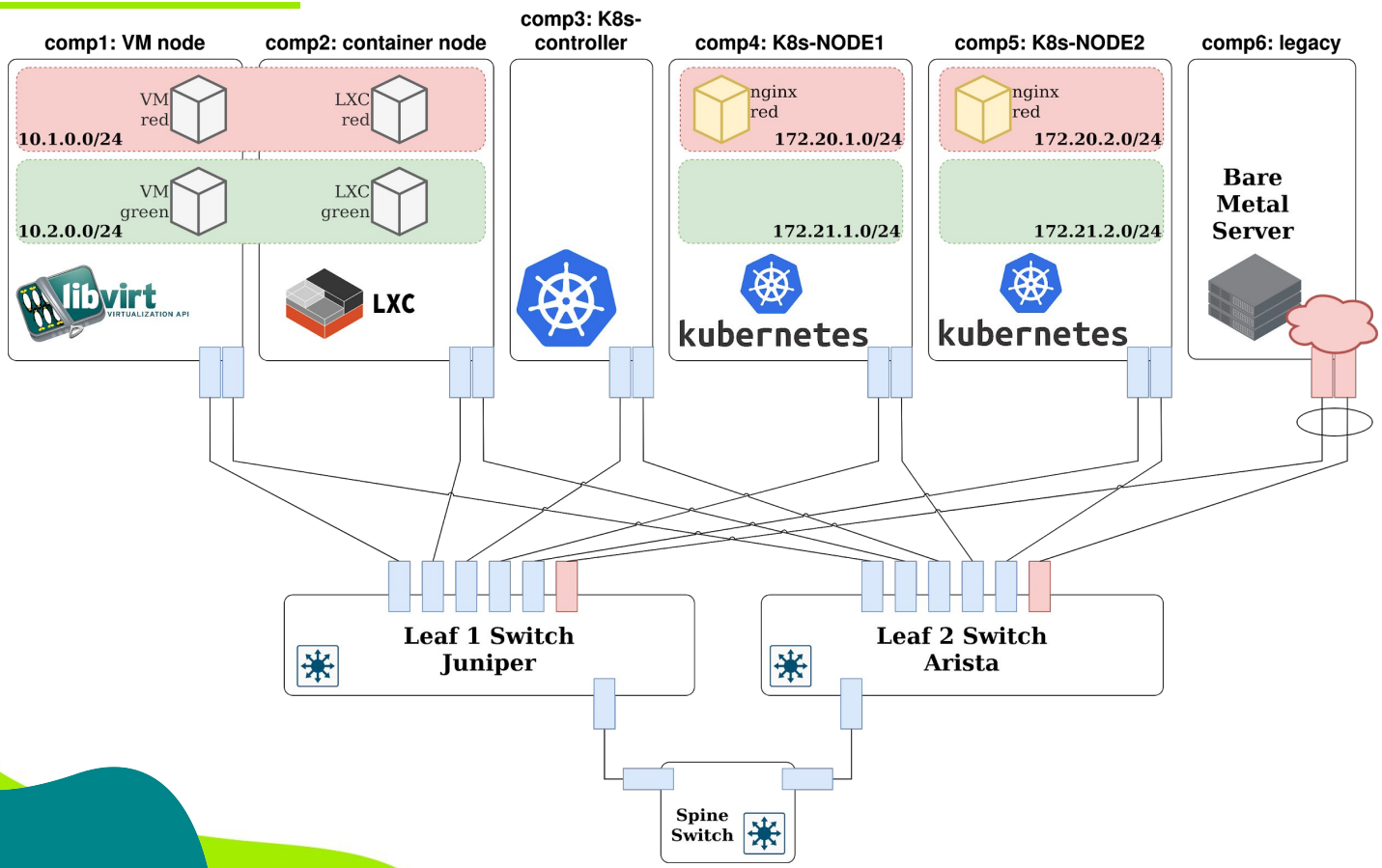
Demo Steps


comp1: VM node
comp2:containernode
comp3: K8s-controller
comp4: K8s-NODE1
comp5: K8s-NODE2
comp6: legacy
VM red
10.1.0.0/24
VM green
10.2.0.0/24
libvirt
VIRTUALIZATION API
LXC
172.20.1.0/24
172.21.1.0/24
kubernetes
172.20.2.0/24
172.21.2.0/24
kubernetes
Bare Metal Server
Leaf 1 Switch Juniper
Leaf 2 Switch Arista
Spine Switch

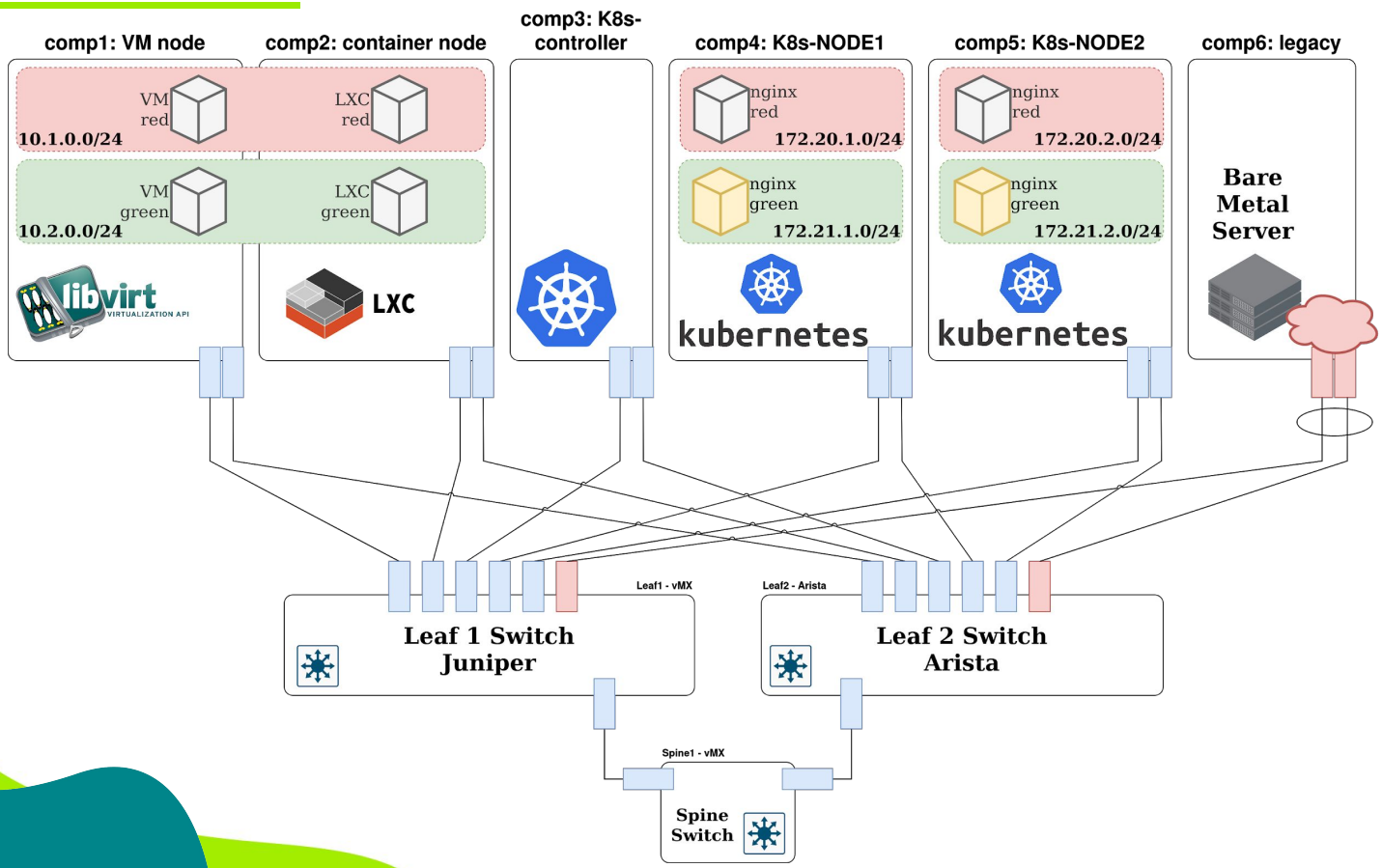
Demo Steps


comp1: VM node
comp2: container node
comp3: K8s-controller
comp4: K8s-NODE1
comp5: K8s-NODE2
comp6: legacy
VM red
10.1.0.0/24
LXC red
VM green
10.2.0.0/24
LXC green
libvirt
VIRTUALIZATION API
LXC
172.20.1.0/24
172.21.1.0/24
kubernetes
172.20.2.0/24
172.21.2.0/24
kubernetes
Bare Metal Server
Leaf 1 Switch Juniper
Leaf 2 Switch Arista
Spine Switch

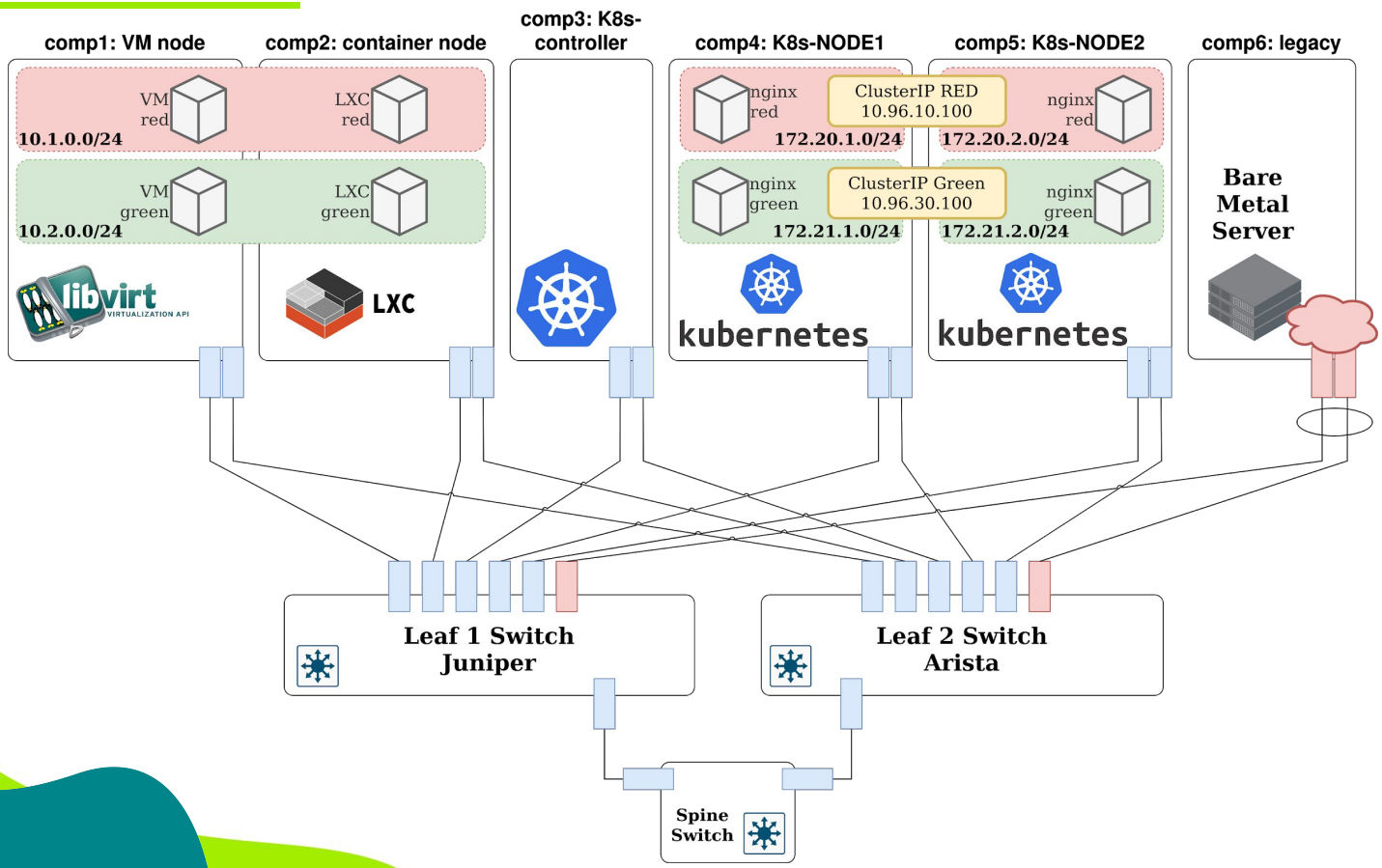
Demo Steps


comp1: VM node
comp2: container node
comp3: K8s-controller
comp4: K8s-NODE1
comp5: K8s-NODE2
comp6: legacy
VM red
10.1.0.0/24
LXC red
VM green
10.2.0.0/24
LXC green
libvirt
VIRTUALIZATION API
LXC
nginx red
172.20.1.0/24
172.21.1.0/24
kubernetes
nginx red
172.20.2.0/24
172.21.2.0/24
kubernetes
Bare Metal Server
Leaf 1 Switch Juniper
Leaf 2 Switch Arista
Spine Switch

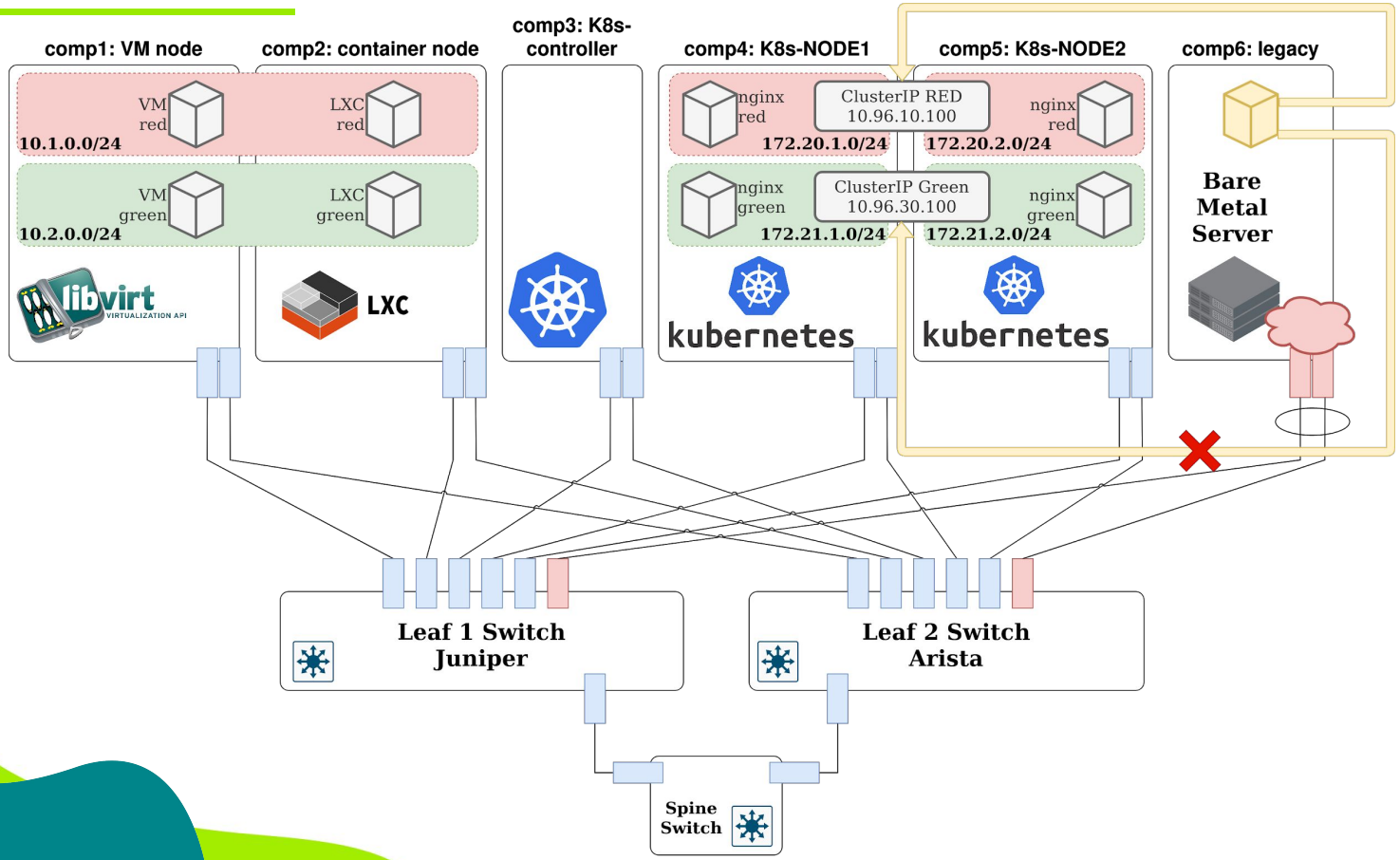
Demo Steps


comp1: VM node
comp2: container node
comp3: K8s-controller
comp4: K8s-NODE1
comp5: K8s-NODE2
comp6: legacy
VM red
10.1.0.0/24
LXC red
VM green
10.2.0.0/24
LXC green
libvirt
VIRTUALIZATION API
LXC
nginx red
172.20.1.0/24
nginx green
172.21.1.0/24
kubernetes
nginx red
172.20.2.0/24
nginx green
172.21.2.0/24
kubernetes
Bare Metal Server
Leaf1 - vMX
Leaf 1 Switch Juniper
Leaf2 - Arista
Leaf 2 Switch Arista
Spine1 - vMX
Spine Switch

Demo Steps


comp1: VM node
comp2: container node
comp3: K8s-controller
comp4: K8s-NODE1
comp5: K8s-NODE2
comp6: legacy
VM red
10.1.0.0/24
VM green
10.2.0.0/24
LXC red
LXC green
libvirt
VIRTUALIZATION API
LXC
nginx red
ClusterIP RED
10.96.10.100
172.20.1.0/24
172.20.2.0/24
nginx green
ClusterIP Green
10.96.30.100
172.21.1.0/24
172.21.2.0/24
kubernetes
kubernetes
Bare Metal Server
Leaf 1 Switch Juniper
Leaf 2 Switch Arista
Spine Switch

Demo Steps


comp1: VM node
comp2: container node
comp3: K8s-controller
comp4: K8s-NODE1
comp5: K8s-NODE2
comp6: legacy
VM red
10.1.0.0/24
VM green
10.2.0.0/24
LXC red
LXC green
libvirt
VIRTUALIZATION API
LXC
nginx red
ClusterIP RED
10.96.10.100
172.20.1.0/24
172.20.2.0/24
nginx green
ClusterIP Green
10.96.30.100
172.21.1.0/24
172.21.2.0/24
kubernetes
kubernetes
Bare Metal Server
Leaf 1 Switch Juniper
Leaf 2 Switch Arista
Spine Switch

Pros & cons of the proposed solution

Pros of using EVPN

- Ability to transfer Layer 2 traffic over Layer 3 connections
  - Includes DCI links, connections to public Clouds and Internet in general (covered in next webinar)
- Interoperability between hardware vendors and software routing daemons
  - No dependence on hardware or software from a single supplier
  - Perfectly OK to have multi-vendor Data Center
- Can interconnect different kinds of resources - BMS/Network appliance, VRFs on Linux, VMs, Containers, etc
  - Very useful for seamless migrating applications between platforms
  - Enables heterogeneous Data Center
- Extra mechanisms built-in
  - Active/active connectivity, can have more than 2 links to legacy server
  - Proxy ARP
  - MAC mobility
- Most switches in DC do not need to support EVPN or VXLAN (if FRR is used on servers)

Pros & cons of the proposed solution

Cons of EVPN

- Advanced solution, requires good networking knowledge
  - Can be simplified through scripts/playbooks/etc.
- Relatively new protocol
  - Stable, but should expect minor bugs in some implementations
- Slightly lower throughput
  - Extra headers lead to lower throughput (effect can be minimized by using Jumbo Frames)
  - Extra CPU utilization on servers (effect can be minimized by using NICs with VXLAN offload)

Problems encountered

- Each EVPN node sending L3 traffic towards Juniper switch must also advertise at least one unique Type-5 route itself
- On virtual Arista MTU had to be increased using OS shell for **fwd0** and **vx1** interfaces for Jumbo Frames to flow through VXLAN tunnels
- To use veth interfaces for route leaking between GRT and VRFs the route table priorities had to be changed and the **local RT** had to be placed after VRF route tables (l3mdev-table) :

```
root@k8s:~# ip rule
0:        from all lookup local ← This rule should be moved to 32765
1000:     from all lookup [l3mdev-table]
32766:    from all lookup main
32767:    from all lookup default
```

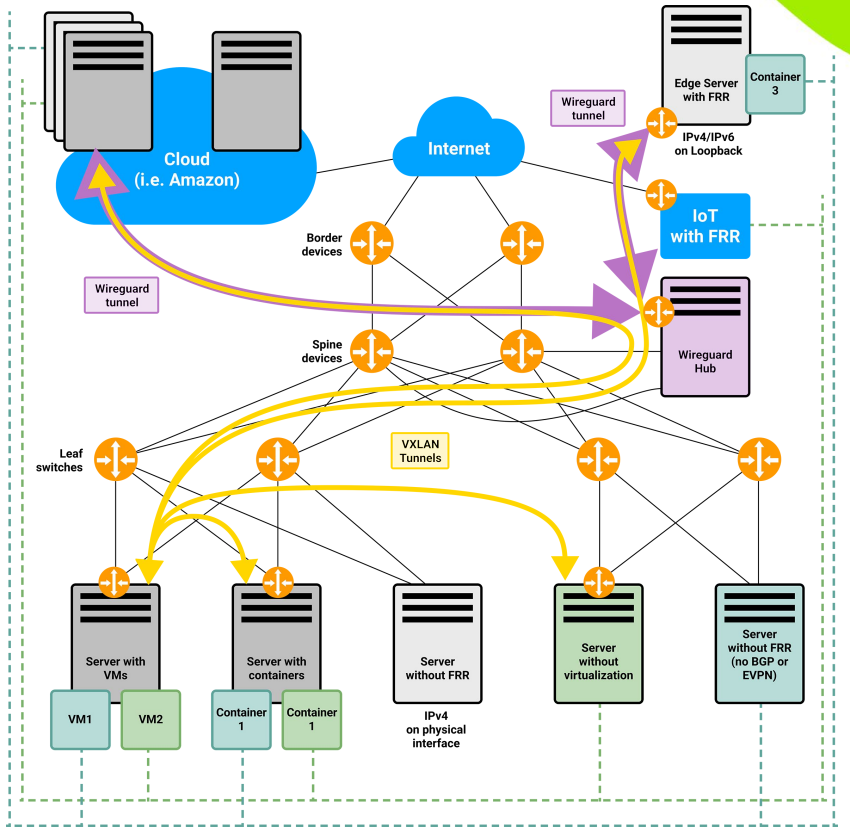
- Due to ECMP load balancing and statefulnes of iptables `--masquerade-all` had to be enabled on kube-proxy
- `rp_filter` had to be disabled on **comp2** or `advertise-svi-ip` had to be enabled in FRR on **comp1** for DHCP to work
- Bridge hairpinning had to be enabled on L3 bridges on K8s nodes for ClusterIP load balancing
- A few FRR issues (prompt response from the FRR team, fix often came within 48 hours)

Summary

- Ethernet VPN is a great tool for Layer 2 connectivity in the modern DC
- Allows for flexible deployments with multiple types of resources, provided by multiple vendors
- Priceless when migrating workloads (i.e. from BMS to VMs/containers)
- Very good automation possibilities (both on networking devices and on servers)
- Multitenancy for multiple services and tenants
- Scalable solution, can support a very large number of overlay networks

Modern DC Series - Webinar 3

- VXLAN routing between two overlay networks
- Overlay-to-underlay gateway
- Connecting two or more DCs together
- Extending Layer 2 connectivity to public clouds
- Secure tunnels for Edge Computing and IoT devices
- Branch office EVPN



Questions & Answers

Configurations, source code, topology, demo recording available at:

https://github.com/codilime/modern-dc

Thank you

codilime®
CREATING VALUE

2100 Geng Road, Suite 210
Palo Alto, CA 94303
United States of America
+1 650 285 2458

Krancowa 5
02-493, Warsaw
Poland
+48 22 389 51 00

al. Grunwaldzka 472B (OBC)
80-309 Gdansk
Poland
+48 575 700 785

contact@codilime.com

SDN & NFV ▢ Cloud native & Multicloud ▢ Software Engineering ▢ UX / UI ▢ DevOps

External sources

While preparing this demo we found the following sources/links helpful:

CNI plugin: https://github.com/coreos/kube-namespace-cni
FRR slack: https://app.slack.com/client/