Configure Easy Virtual Network

Contents

Introduction

Prerequisites

- Requirements
- Components Used

Background Information

Configure

- Network Diagram
- Configure EVN
- Tune the VNET Trunk
  - Trunk List
  - Per-VRF Trunk Attributes
  - Per-Link VNET Tags

Verify

Troubleshoot

Related Information

Introduction

This document describes the Easy Virtual Network (EVN) feature, which provides a simple-to-configure virtualization mechanism in campus networks.

Prerequisites

Requirements

There are no specific requirements for this document.

Components Used

The information in this document is based on these hardware and software versions:

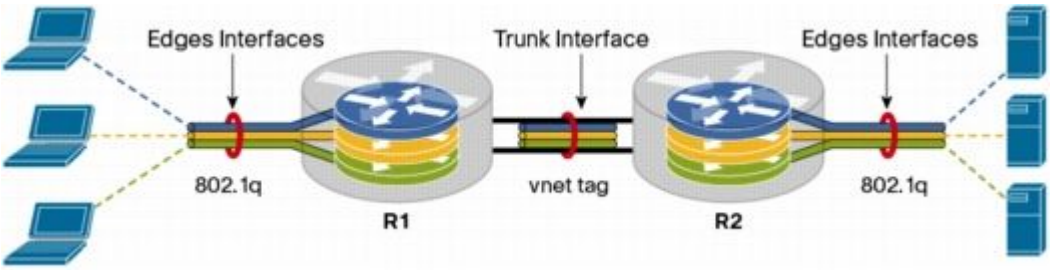
- Cisco Catalyst 6000 (Cat6k) Series switches that run software Version 15.0(1)SY1
- Cisco 1000 Series Aggregated Services Routers (ASR1000) that run software Version 3.2s
- Cisco 3925 and 3945 Series Integrated Services Routers that run Cisco IOS® Versions 15.3(2)T and later
- Cisco Catalyst 4500 (Cat4500) and 4900 (Cat4900) Series switches that run software Version

15.1(1)SG

The information in this document was created from the devices in a specific lab environment. All of the devices used in this document started with a cleared (default) configuration. If your network is live, ensure that you understand the potential impact of any command.

Background Information

EVN leverages current technologies such as Virtual Routing and Forwarding-Lite (VRF-Lite) and dot1q encapsulation and does not introduce any new protocol. Here is an overview of the EVN feature:



- The EVN feature uses VRF-Lite in order to create several (up to 32) routing contexts.
- The connectivity within the Virtual Routing and Forwarding (VRF) between Layer3 devices is ensured via Virtual Network (VNET) trunks.
- The VNET trunks are regular dot1q trunks.
- Configure each VRF that must be transported across the VNET trunks with a VNET tag.
- Each VNET tag equals a dot1q tag.
- **The dot1q subinterfaces are automatically created and hidden.**
- **The configuration of the main interface is inherited by all (hidden) subinterfaces**.
- Use separate instances of routing protocols in each VRF over the VNET trunks in order to advertise prefix reachability.
- Dynamic route leaking between VRFs (opposed to static routes) is allowed without the use of Border Gateway Protocol (BGP).
- The feature is supported for IPv4 and IPv6.

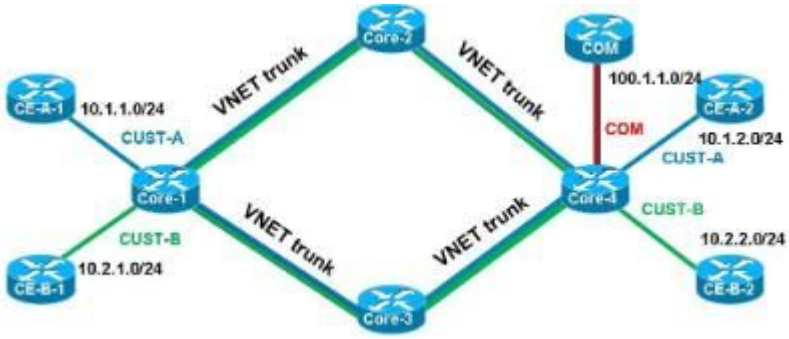
Configure

Use the information that is described in this section in order to configure the EVN feature.

Note: Use the Command Lookup Tool (registered customers only) in order to obtain more information on the commands used in this section.

Network Diagram

This network setup is used in order to illustrate the EVN configuration and show commands:



Here are some important notes about this setup:

- Two VRFs are defined (**CUST-A** and **CUST-B**) that are carried out from the core of the network through VNET trunks.
- Open Shortest Path First (OSPF) is used in the VRFs in order to advertise reachability.
- The VRF COM hosts a common server (**100.1.1.100**) that must be reachable from both VRF CUST-A and CUST-B.
- The image that is used is **i86bi_linux-adventerprisek9-ms.153-1.S**.

Configure EVN

Complete these steps in order to configure the EVN feature:

1. Configure the VRF definition:

```
vrf definition [name]
vnet tag [2-4094]
!
address-family ipv4|ipv6
exit-address-family
!
```

Important notes about this configuration:

- Use tags in the range of 2 to 1,000. Do not use the reserved VLANs 1,001 through 1,005. Use extended VLANs 1,006 through 4,094 if needed.
- Do not allow the VNET tag to be used by a current VLAN.
- Ensure the VNET tags are the same on all devices for any given VRF.
- Configure the **address-family ipv4|ipv6** to activate the VRF in the related AF.
- There is no need to define a Route Direction (RD) because EVN does not use BGP.

With this setup, the VRFs are defined on all 4x core routers. For example, on CORE-1:

```
vrf definition CUST-A
   vnet tag 100
   !
   address-family ipv4
   exit-address-family
  vrf definition CUST-B
   vnet tag 200
   !
   address-family ipv4
   exit-address-family
```

Use the same VNET tag on all of the routers for these VRFs. On CORE-4, VRF COM does not require a VNET tag. The goal is to keep that VRF local on CORE-4 and configure the leaking and redistribution in order to provide access to the common server from CUST-A and CUST-B.

Enter this command in order to check various VNET counters:

<#root>

CORE-1#

show vnet counters

```
  Maximum number of VNETs supported: 32
  Current number of VNETs configured: 2
  Current number of VNET trunk interfaces: 2
  Current number of VNET subinterfaces: 4
  Current number of VNET forwarding interfaces: 6
  CORE-1#
```

2. Configure the VNET trunk:

```
interface GigabitEthernetx/x
  vnet trunk
  ip address x.x.x.x y.y.y.y
  ...
```

Here are some important notes about this configuration:

- The **vnet trunk** command creates as many dot1q subinterfaces as the number of VRFs that are defined with a VNET tag.
- The **vnet trunk** command cannot coexist with some manually-configured subinterfaces on the same physical interface.
- This configuration is allowed on routed interfaces (not switch ports), physical and portchannel.
- The IP addresses (and other commands) that are applied on the physical interface are inherited by the subinterfaces.
- The subinterfaces for all of the VRFs use the same IP address.

With this setup, there are two VNET VRFs, so two subinterfaces are automatically created on the interface that is configured as the VNET trunk. You can enter the **show derived-config** command in order to view the hidden configuration that is automatically created:

Here is the configuration that runs currently:

```
<#root>

CORE-1#

show run | s Ethernet0/0


interface Ethernet0/0
 vnet trunk
 ip address 192.168.1.1 255.255.255.252
 !
CORE-1#
```

Here is the derived configuration:

```
<#root>

CORE-1#

show derived-config | s Ethernet0/0


interface Ethernet0/0
 vnet trunk
 ip address 192.168.1.1 255.255.255.252
Interface Ethernet0/0.100
 description Subinterface for VNET CUST-A
 encapsulation dot1Q 100
 vrf forwarding CUST-A
 ip address 192.168.1.1 255.255.255.252
interface Ethernet0/0.200
 description Subinterface for VNET CUST-B
 encapsulation dot1Q 200
 vrf forwarding CUST-B
 ip address 192.168.1.1 255.255.255.252
CORE-1#
```

As shown, all of the subinterfaces inherit the IP address of the main interface.

3. Assign edge (sub)interfaces to the VRFs. In order to assign an interface or subinterface to a VNET VRF, use the same procedure as that used in order to assign a VRF normally:

```
interface GigabitEthernet x/x.y
 vrf forwarding [name]
  ip address x.x.x.x y.y.y.y
  ...
```

With this setup, the configuration is applied on CORE-1 and CORE-4. Here is an example for CORE-

4:

```
interface Ethernet2/0
   vrf forwarding CUST-A
   ip address 10.1.2.1 255.255.255.0
  !
  interface Ethernet3/0
   vrf forwarding CUST-B
   ip address 10.2.2.1 255.255.255.0
  !
  interface Ethernet4/0
   vrf forwarding COM
   ip address 100.1.1.1 255.255.255.0
```

4. Configure routing protocols for each VRF (this is not specific to EVN or VNET):

```
<#root>

router ospf x vrf [name]

network x.x.x.x y.y.y.y area x

  ...
```

Note: This configuration must include the VNET trunk addresses as well as the edge interface addresses.

With this setup, two OSPF processes are defined, one per VRF:

```
<#root>

CORE-1#

show run | s router os

  router ospf 1 vrf CUST-A
   network 10.1.1.0 0.0.0.255 area 0
   network 192.168.1.0 0.0.0.255 area 0
  router ospf 2 vrf CUST-B
   network 10.2.1.0 0.0.0.255 area 0
   network 192.168.1.0 0.0.0.255 area 0
  CORE-1#
```

You can enter the routing-context mode in order to view the information that is related to a specific VRF without the VRF specifications in each command:

```
<#root>

CORE-1#

routing-context vrf CUST-A


  CORE-1%CUST-A#
  CORE-1%CUST-A#

show ip protocols


  *** IP Routing is NSF aware ***

  Routing Protocol is "ospf 1"
    Outgoing update filter list for all interfaces is not set
    Incoming update filter list for all interfaces is not set
    Router ID 192.168.1.13
    It is an area border router
    Number of areas in this router is 1. 1 normal 0 stub 0 nssa
    Maximum path: 4
    Routing for Networks:
      10.1.1.0 0.0.0.255 area 0
      192.168.1.0 0.0.0.255 area 0
    Routing Information Sources:
      Gateway         Distance      Last Update
      192.168.1.9          110      1d00h
      192.168.1.14         110      1d00h
    Distance: (default is 110)

  CORE-1%CUST-A#

  CORE-1%CUST-A#

show ip os neighbor



Neighbor ID     Pri   State         Dead Time   Address         Interface
192.168.1.14      1   FULL/DR       00:00:30    192.168.1.14    Ethernet1/0.100
192.168.1.5       1   FULL/BDR      00:00:37    192.168.1.2     Ethernet0/0.100
10.1.1.2          1   FULL/BDR      00:00:33    10.1.1.2        Ethernet2/0
CORE-1%CUST-A#
```

Note: The **show ip protocols** command output displays only the information that is related to the selected VRF.

When you view the Routing Information Base (RIB) for both VRFs, you can verify the remote subnet via the two VNET trunks:

```
<#root>

CORE-1%CUST-A#

show ip route 10.1.2.0
```

```
  Routing Table: CUST-A
  Routing entry for 10.1.2.0/24
    Known via "ospf 1", distance 110, metric 30, type intra area
    Last update from 192.168.1.2 on Ethernet0/0.100, 1d00h ago
    Routing Descriptor Blocks:
    * 192.168.1.14, from 192.168.1.9, 1d00h ago, via Ethernet1/0.100
        Route metric is 30, traffic share count is 1
      192.168.1.2, from 192.168.1.9, 1d00h ago, via Ethernet0/0.100
        Route metric is 30, traffic share count is 1
  CORE-1%CUST-A#
  CORE-1%CUST-A#routing-context vrf CUST-B
  CORE-1%CUST-B#
  CORE-1%CUST-B#
```

show ip route 10.2.2.0

```
  Routing Table: CUST-B
  Routing entry for 10.2.2.0/24
    Known via "ospf 2", distance 110, metric 30, type intra area
    Last update from 192.168.1.2 on Ethernet0/0.200, 1d00h ago
    Routing Descriptor Blocks:
    * 192.168.1.14, from 192.168.1.6, 1d00h ago, via Ethernet1/0.200
        Route metric is 30, traffic share count is 1
      192.168.1.2, from 192.168.1.6, 1d00h ago, via Ethernet0/0.200
        Route metric is 30, traffic share count is 1
  CORE-1%CUST-B#
  CORE-1%CUST-B#exit
  CORE-1#
  CORE-1#
```

5. Determine the route leaking between the VRFs. Route leaking is performed via route replication. For example, some routes in a VRF can be made available for another VRF:

<#root>

vrf definition

VRF-X

address-family ipv4|ipv6
route-replicate from vrf

VRF-Y

unicast|

multicast

[

route-origin

] [route-map [name]]

Here are some important notes about this configuration:

- The RIB for **VRF-X** has access to the selected routes, based on command parameters from **VRF-Y**.
- The replicated routes in **VRF-X** are marked with a **[+]** flag.
- The **multicast** option allows the use of routes from another VRF for Reverse Path Forwarding (RPF).
- The **route-origin** can have one of these values:
  - **all**
  - **bgp**
  - **connected**
  - **eigrp**
  - **isis**
  - **mobile**
  - **odr**
  - **ospf**
  - **rip**
  - **static**

Despite the name, route replication does not replicate or duplicate routes. This is normal leaking through BGP common RT and does not consume extra memory.

With this setup, route leaking is used on CORE-4 in order to provide access from CUST-A and CUST-B to COM (and vice-versa):

<#root>

```
vrf definition CUST-A
  address-family ipv4
     route-replicate from vrf COM unicast connected
!
vrf definition CUST-B
  address-family ipv4
     route-replicate from vrf COM unicast connected
!
vrf definition COM
  address-family ipv4
     route-replicate from vrf CUST-A unicast ospf 1 route-map USERS
     route-replicate from vrf CUST-B unicast ospf 2 route-map USERS
!
route-map USERS permit 10
  match ip address prefix-list USER-SUBNETS
!
ip prefix-list USER-SUBNETS seq 5 permit 10.0.0.0/8 le 32

CORE-4#
```

show ip route vrf CUST-A

```
Routing Table: COM
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area,* - candidate default, U - per-user static route
       o - ODR, P - periodic downloaded static route, H - NHRP, l - LISP
       a - application route
```

```
+ - replicated route, % - next hop override


...
       10.0.0.0/8 is variably subnetted, 3 subnets, 2 masks
O        10.1.1.0/24 [110/30] via 192.168.1.10, 3d19h, Ethernet1/0.100
           [110/30] via 192.168.1.5, 3d19h, Ethernet0/0.100
       100.0.0.0/8 is variably subnetted, 2 subnets, 2 masks

C   +    100.1.1.0/24 is directly connected (COM), Ethernet4/0



CORE-4#

show ip route vrf CUST-B


...
       10.0.0.0/8 is variably subnetted, 3 subnets, 2 masks
O        10.2.1.0/24 [110/30] via 192.168.1.10, 1d00h, Ethernet1/0.200
           [110/30] via 192.168.1.5, 1d00h, Ethernet0/0.200
       100.0.0.0/8 is variably subnetted, 2 subnets, 2 masks

C   +    100.1.1.0/24 is directly connected (COM), Ethernet4/0



CORE-4#

show ip route vrf COM


...
       10.0.0.0/24 is subnetted, 2 subnets

O   +    10.1.1.0 [110/30] via 192.168.1.10 (CUST-A), 3d19h, Ethernet1/0.100
           [110/30] via 192.168.1.5 (CUST-A), 3d19h, Ethernet0/0.100
O   +    10.2.1.0 [110/30] via 192.168.1.10 (CUST-B), 1d00h, Ethernet1/0.200
           [110/30] via 192.168.1.5 (CUST-B), 1d00h, Ethernet0/0.200


       100.0.0.0/8 is variably subnetted, 2 subnets, 2 masks
C        100.1.1.0/24 is directly connected, Ethernet4/0
```

At this point, the replicated routes are not propagated in the Interior Gateway Protocol (IGP), so only CE-A-2 and CE-B-2 have access to COM service (100.1.1.100), not CE-A-1 and CE-B-1.

You can also use route leaking from or to a global table:

<#root>

```
vrf definition VRF-X
 address-family ipv4
  route-replicate from vrf >

global
```

```
 unicast|multicast [route-origin]
   [route-map [name]]
  exit-address-family
 !
 exit
!
```

global-address-family ipv4 unicast

```
 route-replicate from vrf [vrf-name] unicast|multicast [route-origin]
  [route-map [name]]
```

6. Define the route leaking propagation. The leaked routes are not duplicated in the target VRF RIB. In other words, they are not a part of the target VRF RIB. Normal redistribution between router processes does not work, so you must explicitly define the VRF connection of the RIB to which the route belongs:

<#root>

```
router ospf x vrf VRF-X
  redistribute
```

vrf VRF-Y

```
 [route-origin] [route-map [name]]
```

The leaked routes from VRF-Y are redistributed in the OSPF process that runs in VRF-X. Here is an example on CORE-4:

```
router ospf 1 vrf CUST-A
   redistribute vrf COM connected subnets route-map CON-2-OSPF
!
route-map CON-2-OSPF permit 10
 match ip address prefix-list COM
!
ip prefix-list COM seq 5 permit 100.1.1.0/24
```

The route-map is not needed in this case, since there is only one connected route in VRF COM. There is now reachability to COM service (100.1.1.100) from CE-A-1 and CE-B-1:

<#root>

```
CE-A-1#
```

ping 100.1.1.100

```
  Type escape sequence to abort.
  Sending 5, 100-byte ICMP Echos to 100.1.1.100, timeout is 2 seconds:
  !!!!!
  Success rate is 100 percent (5/5), round-trip min/avg/max = 1/1/1 ms
  CE-A-1#
```

```
CE-B-1#
ping 100.1.1.100

  Type escape sequence to abort.
  Sending 5, 100-byte ICMP Echos to 100.1.1.100, timeout is 2 seconds:
  !!!!!
  Success rate is 100 percent (5/5), round-trip min/avg/max = 1/1/1 ms
  CE-B-1#
```

Tune the VNET Trunk

This section provides information that you can use in order to tune the VNET trunk.

Trunk List

By default, all of the VRFs that are configured with a VNET tag are allowed on all of the VNET trunks. A trunk list allows you to specify the list of authorized VRFs on the VNET trunk:

<#root>

```
vrf list [list-name]
```

member [vrf-name]

```
!
interface GigabitEthernetx/x
  vnet trunk
```

list [list-name]

Note: Use one line per allowed VRF.

As an example, CORE-1 is tuned for the VRF CUST-B on the VNET trunk between CORE-1 and CORE-2:

```
vrf list TEST
 member CUST-A
!
interface ethernet0/0
 vnet trunk list TEST
```

As shown, the OSPF peering for VRF CUST-B across the trunk goes down:

<#root>

```
%OSPF-5-ADJCHG: Process 2, Nbr 192.168.1.2 on Ethernet0/0.200 from FULL to DOWN,
 Neighbor Down: Interface down or detached
```

The subinterface for VRF CUST-B is removed:

<#root>

```
CORE-1#

show derived-config | b Ethernet0/0


  interface Ethernet0/0
   vnet trunk list TEST
   ip address 192.168.1.1 255.255.255.252
  !
  interface Ethernet0/0.100
   description Subinterface for VNET CUST-A
   encapsulation dot1Q 100
   vrf forwarding CUST-A
   ip address 192.168.1.1 255.255.255.252
  !
```

Per-VRF Trunk Attributes

By default, the dot1q subinterfaces inherit the parameters of the physical interface so that the subinterfaces for all of the VRFs have the same attributes (such as cost and authentication). You can tune the trunk parameters per VNET tag:

```
interface GigaEthernetx/x
  vnet trunk
  vnet name VRF-X
   ip ospf cost 100
  vnet name VRF-Y
   ip ospf cost 15
```

You can tune these parameters:

<#root>

```
CORE-1(config-if-vnet)#?
      Interface VNET instance override configuration commands:
```

bandwidth

```
     Set bandwidth informational parameter
        default       Set a command to its defaults
```

delay

```
         Specify interface throughput delay
        exit-if-vnet  Exit from VNET submode
```

ip

```
            Interface VNET submode Internet Protocol config commands
        no            Negate a command or set its defaults
        vnet          Configure protocol-independent VNET interface options
      CORE-1(config-if-vnet)#

      CORE-1(config-if-vnet)#ip ?
        authentication      authentication subcommands
        bandwidth-percent   Set EIGRP bandwidth limit
        dampening-change    Percent interface metric must change to cause update
        dampening-interval  Time in seconds to check interface metrics
        hello-interval      Configures EIGRP-IPv4 hello interval
        hold-time           Configures EIGRP-IPv4 hold time
        igmp                IGMP interface commands
        mfib                Interface Specific MFIB Control
        multicast           IP multicast interface commands
        next-hop-self       Configures EIGRP-IPv4 next-hop-self
        ospf                OSPF interface commands
        pim                 PIM interface commands
        split-horizon       Perform split horizon
        summary-address     Perform address summarization
        verify              Enable per packet validation

      CORE-1(config-if-vnet)#ip
```

In this example, the the OSPF cost per VRF for CORE-1 is changed, so the CORE-2 path is used for CUST-A and the CORE-3 path for CUST-B (the default cost is **10**

<#root>

```
interface Ethernet0/0
  vnet name CUST-A
    ip ospf cost 8
  !
  vnet name CUST-B
    ip ospf cost 12
  !

CORE-1#
```

show ip route vrf CUST-A 10.1.2.0

```
Routing Table: CUST-A
Routing entry for 10.1.2.0/24
 Known via "ospf 1", distance 110, metric 28, type intra area
 Last update from 192.168.1.2 on Ethernet0/0.100, 00:05:42 ago
 Routing Descriptor Blocks:
 * 192.168.1.2, from 192.168.1.9, 00:05:42 ago, via Ethernet0/0.100
   Route metric is 28, traffic share count is 1
CORE-1#

CORE-1#
```

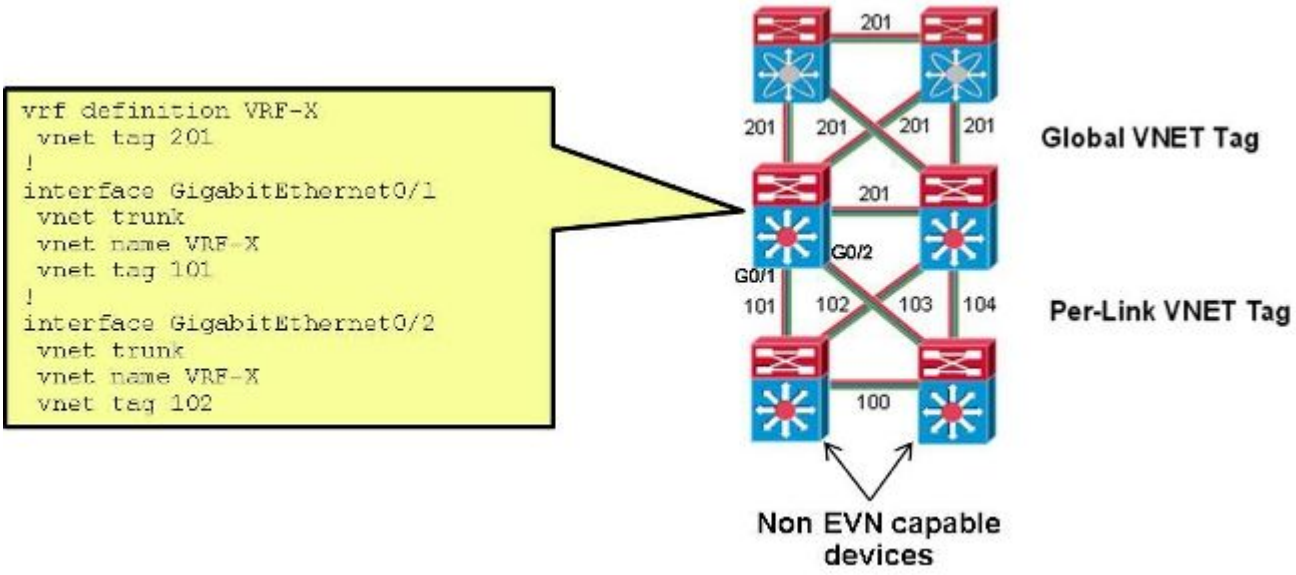
`show ip route vrf CUST-B 10.2.2.0`

```
Routing Table: CUST-B
Routing entry for 10.2.2.0/24
 Known via "ospf 2", distance 110, metric 30, type intra area
 Last update from 192.168.1.14 on Ethernet1/0.200, 00:07:03 ago
 Routing Descriptor Blocks:
 * 192.168.1.14, from 192.168.1.6, 1d18h ago, via Ethernet1/0.200
   Route metric is 30, traffic share count is 1
CORE-1#
```

Per-Link VNET Tags

By default, the VNET tag that is defined in the VRF definition is used for all of the trunks. However, you can use a different VNET tag per trunk.

This example describes a scenario where you are connected to a non-EVN capable device and you use VRF-Lite with a manual trunk, and the global VNET tag is used by another VLAN:



With this setup, the VNET tag that is used on the trunk between CORE-1 and CORE-2 for CUST-A is changed from **100** to **101**:

```
<#root>

interface Ethernet0/0
  vnet name CUST-A
   vnet tag

101
```

After this change occurs on CORE-1, a new subinterface is created:

```
<#root>

CORE-1#

show derived-config | b Ethernet0/0


interface Ethernet0/0
 vnet trunk
 ip address 192.168.1.1 255.255.255.252
!

interface Ethernet0/0.101


 description Subinterface for VNET CUST-A
 encapsulation dot1Q 101
 vrf forwarding CUST-A
 ip address 192.168.1.1 255.255.255.252
!
interface Ethernet0/0.200
 description Subinterface for VNET CUST-B
 encapsulation dot1Q 200
 vrf forwarding CUST-B
 ip address 192.168.1.1 255.255.255.252
```

If this change occurs only on one end, then connectivity is lost in the associated VRF and the OSPF goes down:

```
<#root>

%OSPF-5-ADJCHG: Process 1, Nbr 192.168.1.5 on Ethernet0/0.101 from FULL to DOWN,
 Neighbor Down: Dead timer expired
```

Once the same VNET tag is used on CORE-2, the connectivity is restored and dot1q tag **101** is used on that trunk while **100** is still used on the CORE-1 to CORE-3 trunk:

<#root>

```
%OSPF-5-ADJCHG: Process 1, Nbr 192.168.1.5 on Ethernet0/0.101 from LOADING to
 FULL, Loading Done
```

Verify

There is currently no verification procedure available for this configuration.

Troubleshoot

There is currently no specific troubleshooting information available for this configuration.

Related Information

- **Easy Virtual Network—Simplifying Layer 3 Network Virtualization**
- **Technical Support & Documentation - Cisco Systems**