Engineering
Simplicity

DAY ONE: JUNIPER AMBASSADORS' COOKBOOK FOR 2018

What's on the minds of the Juniper Ambassadors in 2018? It's EVPN, automating with SaltStack, ELS and the EX Series, GRE Tunnels, VLANs, FBF routing, and more EVPN.

By Dan Hearty, Tom Dwyer, Jeff Fry, Michel Tepper, Steve Puluka, Martin Brown, Peter Klimai, and Clay Haynes

DAY ONE: JUNIPER AMBASSADORS' COOKBOOK FOR 2018

The Juniper Ambassador program recognizes and supports its top community members and the generous contributions they make through sharing their knowledge, passion, and expertise on J-Net, Facebook, Twitter, LinkedIn, and other social networks. In their new *Day One* cookbook, the Juniper Ambassadors take on some of the top networking support issues and provide clear-cut solutions and frank discussions on how to keep things running. The recipes in this cookbook provide quick and tested solutions to everyday networking administration issues.

"Networks are moving to the cloud and so are our Juniper Ambassadors. This book is filled with customer solutions using Juniper technologies – from EVPN, to automating Junos with SaltStack. I am grateful for the time and energy our Juniper Ambassadors have devoted to this book. It's yet another proof point that Juniper's focus on customers is paying off."

Rami Rahim, CEO, Juniper Networks

IT'S DAY ONE AND YOU HAVE A JOB TO DO, SO LEARN HOW TO:

- Identify and resolve asymmetric routing problems.
- Provide redundant L3 Gateway in an EVPN-VXLAN fabric with inter-tenant intra-subnet connectivity.
- Use EVPN route type-5 for aggregating multiple host MAC+IP routes for tenants behind a single IP prefix for a given bridge domain.
- Convert your legacy EX4200/EX2200 DHCP configuration over to the new Enhanced Layer 2 Software (ELS).
- Restrict packet sizes that can affect GRE tunnels.
- Extend a VLAN across a Layer 3 WAN without a complete re-architecture of the network
- Implement FBF when the traffic being processed arrives at an interface inside of a virtual router's routing instance.
- Automate certain operational and configuration tasks on your Junos network devices using SaltStack.

ISBN 978-1-941441-79-4
52500
9 781941 441794

Juniper Networks Books are focused on network reliability and efficiency. Peruse the complete library at www.juniper.net/books.

JUNIPer NETWORKS

Day One: Ambassadors' Cookbook for 2018

by Dan Hearty, Tom Dwyer, Jeff Fry, Michel Tepper, Steve Puluka, Martin Brown, Peter Klimai, and Clay Haynes

Table of Contents

© 2018 by Juniper Networks, Inc. All rights reserved.
Juniper Networks, Junos, Steel-Belted Radius, NetScreen, and ScreenOS are registered trademarks of Juniper Networks, Inc. in the United States and other countries. The Juniper Networks Logo, the Junos logo, and JunosE are trademarks of Juniper Networks, Inc. All other trademarks, service marks, registered trademarks, or registered service marks are the property of their respective owners. Juniper Networks assumes no responsibility for any inaccuracies in this document. Juniper Networks reserves the right to change, modify, transfer, or otherwise revise this publication without notice.

© 2018 by Juniper Networks Pvt Ltd. All rights reserved for scripts published within this book.

Script Software License
© 2018 Juniper Networks, Inc. All rights reserved. Licensed under the Juniper Networks Script Software License (the "License"). You may not use this script file except in compliance with the License, which is located at http://www.juniper.net/support/legal/scriptlicense/. Unless required by applicable law or otherwise agreed to in writing by the parties, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

Published by Juniper Networks Books
Authors: Dan Hearty, Tom Dwyer, Jeff Fry, Michel Tepper, Steve Puluka, Martin Brown, Peter Klimai
Technical Editor: Clay Haynes
Editor in Chief: Patrick Ames
Copyeditor and Proofer: Nancy Koerbel
Illustrations: Karen Joice
Ambassador Program Manager: Julie Wider

ISBN: 978-1-941441-79-4 (print)
Printed in the USA by Vervante Corporation.
ISBN: 978-1-941441-80-0 (ebook)
Version History: v1, September, 2018
2 3 4 5 6 7 8 9 10

Visit the entire *Day One* library:
http://www.juniper.net/dayone

Feedback? Comments? Error reports? Email them to dayone@juniper.net.

Contributing Ambassadors

Dan Hearty (Recipe 8 and 9) is a Principal Engineer working for Telent in the UK and is a Juniper Ambassador. He specializes in service provider and data centre technologies with over 10 years' experience. He is JNCIE-SP #2406 and holds a number of other industry certifications.

Tom Dwyer (Recipe 1) is a Principal Engineer leading the Data Center Practice at Nexum Inc, a VAR, MSP and training provider based out of Chicago. He has over 20 years of experience focused on networking, security, and data center technologies. Tom is a Juniper Ambassador and is certified by Juniper as a JNCIE-ENT #424.

Jeff Fry (Recipe 4) works a Senior Consultant at Dimension Data and lives in Northampton, PA with his wife and three sons. He is certified Cisco CCIE R&S 22061, Juniper JNCIE-ENT #567, as well as JNCIP in Service Provider, Security, and Data Center. Jeff started his career over 25 years ago supporting Windows and Unix environments and was bit by the network bug early in his career.

Michel Tepper (Recipe 2 and 6) is a Solutions Architect for Nuvias in the Netherlands and a Juniper Ambassador. Besides doing a lot of presales support he also is a Juniper Networks Certified Instructor on Security, Services provider and Enterprise routing and switching tracks. Working 30+ years in the industry doesn't reduce his enthusiasm for it, or for Juniper in specific. Besides Juniper certifications Michel holds certifications for a number of other leading vendors..

Steve Puluka (Recipe 3) is a Network Architect with DQE Communications in Pittsburgh, PA. He is part of a service provider team that manages a fiber optic Metro Ethernet, Wavelength, and Internet Services network spanning 3k route miles throughout Western Pennsylvania. He holds a BSEET along with a dozen Juniper Certifications in Service Provider, Security, and Design. He also has certification and extensive experience in Microsoft Windows server, along with strong VMWare skills starting with Version 2. He has enjoyed supporting networks for more than 20 years.

Martin Brown (Recipe 5) is a Network Security Engineer for a tier 1 service provider based in the UK and is a Juniper Ambassador. Martin started his career in IT over 20 years ago supporting Macintosh computers and in 1999 earned his first certification by becoming an MCP then an MCSE. In the past six years he has progressed to networking, implementing, and supporting network devices in a number of different environments including airports, retail, warehouses and service providers. His knowledge covers a broad range of network device types and network equipment from most of the major vendors including Cisco, F5, Checkpoint, and of course, Juniper.

(continued)

Peter Klimai (Recipe 7) is a Juniper Ambassador and a Juniper Networks certified instructor working at Poplar Systems, a Juniper-Authorized Education Partner in Russia. He is certified JNCIE-SEC #98, JNCIE-ENT #393, and JNCIE-SP #2253 and has several years of experience supporting Juniper equipment for many small and large companies. He teaches a variety of Juniper classes on a regular basis, beginning with introductory level (such as IJOS) and including advanced (such as AJSEC, JAUT, and NACC). Peter is enthusiastic about network automation using various tools, as well as network function virtualization.

Clay Haynes (Technical Reviewer) is an IT professional with over 10 years of experience working on servers, firewalls, and networking. He currently works at Twitter as a Senior Network Security Engineer and is a Juniper Ambassador. Clay currently holds the JNCIE-SEC #69 and JNCIE-ENT #492 certifications.

Preface

The world we live in has become more interconnected than at any point in history. Consider for a moment the smartphone in your pocket, and how its functions have expanded beyond phone calls and SMS; I use my own phone to read books, watch videos, pay for goods and services, and communicate to people around the globe as if they are in the same room as me. These are activities that were considered unimaginable five years ago.

These applications exist in hosted data centers, cloud environments, or on-premises servers. Regardless of where the application is hosted, we know that behind every application lies a network. While the core foundations of networks have not changed in decades, the hardware has become faster and more compact and its software has increased in complexity to provide new features to meet the demands of customers. The ever-present demand for higher performance from our networks must be balanced with constraints imposed by business, as well as the needs for ensuring security and availability.

The recipes in this cookbook have been authored by many amazing individuals who come from very diverse backgrounds and business verticals. They will help you design, deploy, operate, and maintain networks that are powered by Juniper Networks. And they will help you integrate Juniper Networks with other vendor's solutions. A few of the topics that will be covered in this cookbook include EVPN, MC-LAG, L2 Tunneling, virtualization, automation, working with ELS, and advanced security features.

Enjoy!

Clay Haynes, September 2018
Technical Reviewer & Juniper Ambassador

Recipe 1: Extending Layer 2 Over Layer 3 VPN MPLS with EVPN-VXLAN

by Tom Dwyer

- JunosOSUsed:16.1R7.7(MX)
- JunosOSUsed:15.1X53-D66.8(QFX10002)
- Juniper Platforms General Applicability: vMX, MX, QFX

Temporary topology changes can sometimes create monsters. This recipe tackles a requirement to provide a Layer 2 stretch between two data centers. ACME was just acquired by Phantom Corporation headquartered out of Chicago. They need to migrate a large amount of virtual machines from ACME's Las Vegas data center to Phantom's Chicago data center. Both data centers are using the same MPLS provider, which allows them to quickly provision a Layer 3 VPN between their two networks. This recipe shows how they can use EVPN to solve the Layer 2 stretch without major architectural changes.

Problem

You need to quickly extend a VLAN across a Layer 3 WAN without a complete re-architecture of the network.

Solution

Mergers and acquisitions (M&A) can provide many challenges for a networking staff. Oftentimes IT staff are the last to know about M&A activity, and timelines to integrate the companies involved are usually aggressive. In this case study, this already difficult situation is further escalated when the IT staff finds out that the contract for ACME's data center in Las Vegas is expiring, so it's necessary to migrate compute resources from ACME's Las Vegas data center to Phantom Corporation's Chicago data center.

Since both companies were using the same MPLS provider, adding connectivity across both MPLS networks was made easy by provisioning a new L3VPN. Both companies have deployed Juniper QFX switches in their network that will be leveraged in the solution.

From the high-level design standpoint shown in Figure 1.1, we will create an EVPN-VXLAN connection between the two QFX10002s over our L3VPNs.

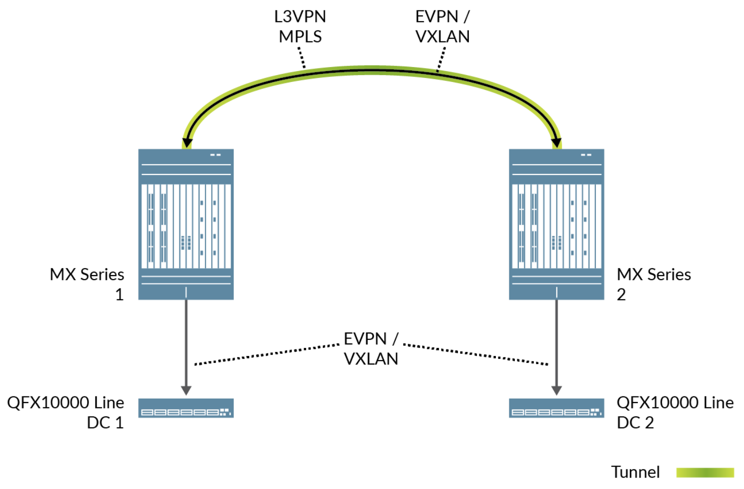


Figure 1.1 *The Solution to Create a EVPN-VXLAN Connection*

The reason EVPN is selected for this purpose is that it requires IP transport to provide connectivity between the virtual tunnel endpoints (VTEPs).

Let's begin to build out the solution. The first step is to build the underlay network. The purpose of the underlay network is to provide connectivity for the VTEPs. The QFX10002 switches are already participating in BGP with each PE router from the MPLS provider to provide connectivity. We will need to modify the BGP export policy to advertise the loopback addresses that are being used for VTEP termination points.

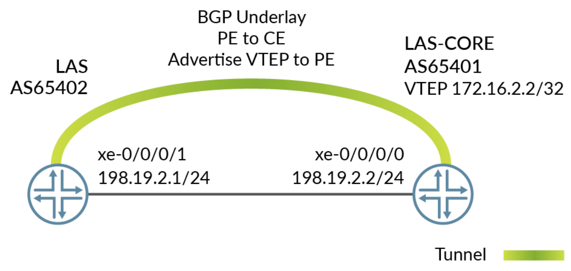


Figure 1.2 *Las Vegas Addresses*

```
admin@LASQFX-1# set interfaces lo0 unit 0 family inet address 172.16.2.2/32

admin@LASQFX-1# set policy-options policy-statement CE-EXPORT term VTEP-EXPORT from protocol direct
admin@LASQFX-1# set policy-options policy-statement CE-EXPORT term VTEP-EXPORT from route-
filter 172.16.2.2/32 exact
admin@LASQFX-1# set policy-options policy-statement CE-EXPORT term VTEP-EXPORT then accept
admin@LASQFX-1# insert policy-options policy-statement CE-EXPORT term VTEP-
EXPORT before term REJECT-ALL


[edit]
admin@LASQFX-1# commit and-quit
commit complete
Exiting configuration mode

admin@LASQFX-1>
```

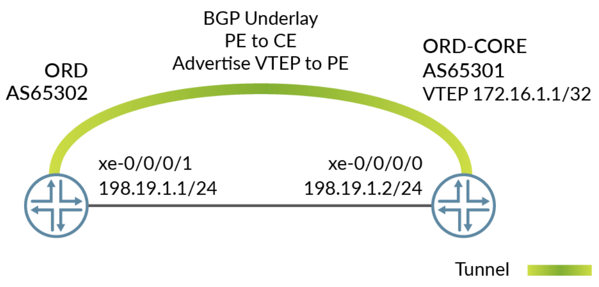


Figure 1.3 Chicago Addresses

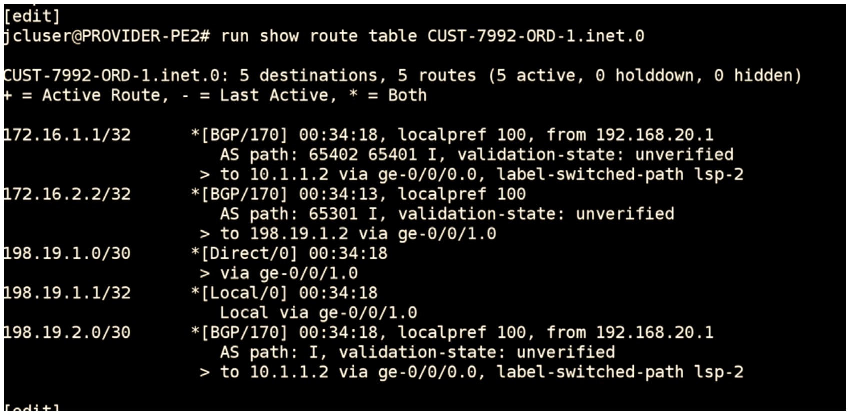
```
admin@ORDQFX-1# set interfaces lo0 unit 0 family inet address 172.16.1.1/32

admin@ORDQFX-1# set policy-options policy-statement CE-EXPORT term VTEP-EXPORT from protocol direct
admin@ORDQFX-1# set policy-options policy-statement CE-EXPORT term VTEP-EXPORT from route-
filter 172.16.1.1/32 exact
admin@ORDQFX-1# set policy-options policy-statement CE-EXPORT term VTEP-EXPORT then accept
admin@ORDQFX-1# insert policy-options policy-statement CE-EXPORT term VTEP-
EXPORT before term REJECT-ALL


[edit]
admin@ORDQFX-1# commit and-quit
commit complete
Exiting configuration mode

admin@ORDQFX-1>
```

Okay, once this is committed we should be able to see the loopbacks at each data center. As shown in Figure 1.3, on the MPLS PE router the loopbacks are being advertised.



```
[edit]
jcluser@PROVIDER-PE2# run show route table CUST-7992-ORD-1.inet.0

CUST-7992-ORD-1.inet.0: 5 destinations, 5 routes (5 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

172.16.1.1/32      *[BGP/170] 00:34:18, localpref 100, from 192.168.20.1
                      AS path: 65402 65401 I, validation-state: unverified
                    > to 10.1.1.2 via ge-0/0/0.0, label-switched-path lsp-2
172.16.2.2/32      *[BGP/170] 00:34:13, localpref 100
                      AS path: 65301 I, validation-state: unverified
                    > to 198.19.1.2 via ge-0/0/1.0
198.19.1.0/30      *[Direct/0] 00:34:18
                    > via ge-0/0/1.0
198.19.1.1/32      *[Local/0] 00:34:18
                      Local via ge-0/0/1.0
198.19.2.0/30      *[BGP/170] 00:34:18, localpref 100, from 192.168.20.1
                      AS path: I, validation-state: unverified
                    > to 10.1.1.2 via ge-0/0/0.0, label-switched-path lsp-2
```

Figure 1.4 Loopbacks Advertised

Now that the underlay has been has been built, it's time to build out the overlay. We will establish an iBGP session between the loopback addresses. Family EVPN signaling is the only Network Layer Reachability Information (NLRI) that is needed to establish the EVPN BGP session:

```
admin@LASQFX-1# set routing-options autonomous-system 65001
admin@LASQFX-1# set protocols bgp group EVPN-OVERLAY type internal
admin@LASQFX-1# set protocols bgp group EVPN-OVERLAY family evpn signaling
admin@LASQFX-1# set protocols bgp group EVPN-OVERLAY neighbor 172.16.1.1
admin@LASQFX-1# set protocols bgp group EVPN-OVERLAY local-address 172.16.2.2


admin@ORDQFX-1# set routing-options autonomous-system 65001
admin@ORDQFX-1# set protocols bgp group EVPN-OVERLAY type internal
admin@ORDQFX-1# set protocols bgp group EVPN-OVERLAY family evpn signaling
admin@ORDQFX-1# set protocols bgp group EVPN-OVERLAY neighbor 172.16.2.2
admin@ORDQFX-1# set protocols bgp group EVPN-OVERLAY local-address 172.16.1.1
```

Let's start by extending one VLAN across the WAN. VLAN 1500 has been provisioned for this purpose and we will provision a `vxlan vni` for it as well.

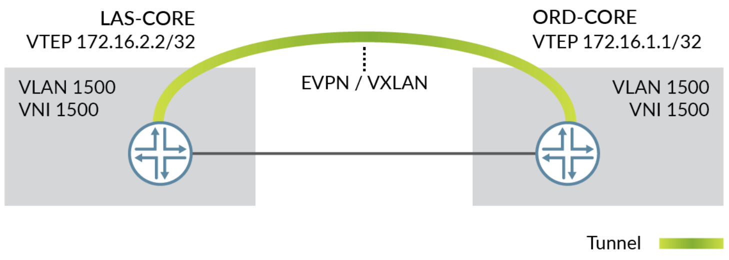


Figure 1.5 Extending VLAN 1500

```
admin@LASQFX-1# set vlans VL1500 vlan-id 1500
admin@LASQFX-1# set vlans VL1500 vxlan vni 1500

admin@ORDQFX-1# set vlans VL1500 vlan-id 1500
admin@ORDQFX-1# set vlans VL1500 vxlan vni 1500
```

NOTE In our example the VNI and VLAN-ID have kept the same numerical values. For simplicities sake, keeping both of them associated with the same value is consistent with best practices that have been around with logical associations. VXLAN has 24 bits and can scale beyond VLAN scalability. This type of scale is beyond the scope of this recipe.

Now let's modify the switch options to source the interface for the VTEP, using lo0.0 for the VTEP, and configuring a route distinguisher and a route target:

```
admin@LASQFX-1# set switch-options vtep-source-interface lo0.0
admin@LASQFX-1# set switch-options route-distinguisher 172.16.2.2:1
admin@LASQFX-1# set switch-options vrf-target target:7777:7777
admin@LASQFX-1# set switch-options vrf-import EVPN-IMPORT


admin@ORDQFX-1# set switch-options vtep-source-interface lo0.0
admin@ORDQFX-1# set switch-options route-distinguisher 172.16.1.1:1
admin@ORDQFX-1# set switch-options vrf-target target:7777:7777
admin@ORDQFX-1# set switch-options vrf-import EVPN-IMPORT
```

Create Layer 3 interfaces for the VLAN:

```
admin@ORDQFX-1# set interfaces irb unit 1500 family inet address 10.1.50.2/24
admin@ORDQFX-1# set vlans VL1500 l3-interface irb.1500

admin@LASQFX-1# set interfaces irb unit 1500 family inet address 10.1.50.3/24
admin@LASQFX-1# set vlans VL1500 l3-interface irb.1500
```

NOTE Not all VTEPs are built the same. Some devices (QFX5100, QFX5200) can only support a Layer 2 VTEP due to the ASICs that power them. They cannot route EVPN inter-VXLAN traffic. This is important to know about when designing where routing will occur in an EVPN network. In a mixed fabric of QFX5100 and QFX10002 devices, the QFX10002 would provide routing at the spine.

Add the `vni` for VLAN 1500 to the EVPN protocol and add a target for export:

```
admin@LASQFX-1# set protocols evpn encapsultation vxlan
admin@LASQFX-1# set protocols evpn extended-vni-list 1500
admin@LASQFX-1# set protocols evpn multicast-mode ingress-replication
admin@LASQFX-1# set protocols evpn default-gateway no-gateway-community
admin@LASQFX-1# set protocols evpn vni-options 1500 export target:1:1500

admin@ORDQFX-1# set protocols evpn encapsultation vxlan
admin@ORDQFX-1# set protocols evpn extended-vni-list 1500
admin@ORDQFX-1# set protocols evpn multicast-mode ingress-replication
admin@ORDQFX-1# set protocols evpn default-gateway no-gateway-community
admin@ORDQFX-1# set protocols evpn vni-options 1500 export target:1:1500
```

Let's now define the community values for the Type 1 routes defined by the route target statement under switch options, and also, create a community for the Type 2 route import:

```
admin@ORDQFX-1# set policy-options community evpn-type1 members target:7777:7777
admin@ORDQFX-1# set policy-options community vni-1500 members target:1:1500

admin@LASQFX-1# set policy-options community evpn-type1 members target:7777:7777
admin@LASQFX-1# set policy-options community vni-1500 members target:1:1500
```

Next, define the import policy for EVPN, which will import routes into the default switch.evpn.0 table from the bgp.evpn.0 table:

```
admin@ORDQFX-1# set policy-options policy-statement EVPN-IMPORT term EVPN-TYPE1 from community evpn-
type1
admin@ORDQFX-1# set policy-options policy-statement EVPN-IMPORT term EVPN-TYPE1 then accept
admin@ORDQFX-1# set policy-options policy-statement EVPN-IMPORT term VNI-1500 from community vni-
1500
admin@ORDQFX-1# set policy-options policy-statement EVPN-IMPORT term VNI-1500 then accept
admin@ORDQFX-1# set policy-options policy-statement EVPN-IMPORT term default-reject
admin@ORDQFX-1# set policy-options policy-statement EVPN-IMPORT term  default-reject then reject

admin@LASQFX-1# set policy-options policy-statement EVPN-IMPORT term EVPN-TYPE1 from community evpn-
type1
admin@LASQFX-1# set policy-options policy-statement EVPN-IMPORT term EVPN-TYPE1 then accept
admin@LASQFX-1# set policy-options policy-statement EVPN-IMPORT term VNI-1500 from community vni-
1500
admin@LASQFX-1# set policy-options policy-statement EVPN-IMPORT term VNI-1500 then accept
admin@LASQFX-1# set policy-options policy-statement EVPN-IMPORT term default-reject
admin@ORDQFX-1# set policy-options policy-statement EVPN-IMPORT term default-reject then reject

[edit]
admin@ORDQFX-1# commit and-quit
commit complete
Exiting configuration mode

admin@ORDQFX-1>configure


edit]
admin@LASQFX-1# commit and-quit
commit complete
Exiting configuration mode
```

After commiting, this configuration should build out the topology shown in Figure 1.6, creating an EVPN-VXLAN connection across the L3VPN network.

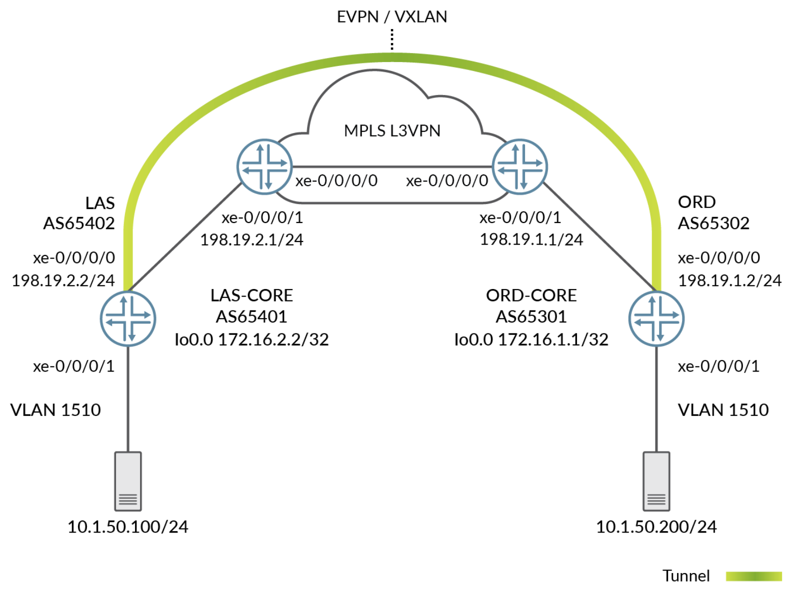


Figure 1.6 *Connection Across the L3VPN Network*

Now let’s verify that our EVPN is working:

Let’s first validate that the overlay is up and that EVPN is communicating. BGP is established between the loopbacks. You can see that the EVPN family and NRLI are present:

```
admin@LASQFX-1> show bgp neighbor 172.16.1.1
Peer: 172.16.1.1+54265 AS 65001 Local: 172.16.2.2+179 AS 65001
  Group: EVPN-OVERLAY           Routing-Instance: master
  Forwarding routing-instance: master
  Type: Internal    State: Established    Flags: <Sync>
  Last State: OpenConfirm   Last Event: RecvKeepAlive
  Last Error: None
  Options: <Preference LocalAddress LogUpDown AddressFamily Rib-group Refresh>
  Address families configured: evpn
  Local Address: 172.16.2.2 Holdtime: 90 Preference: 170


  admin@ORDQFX-1> show bgp neighbor 172.16.2.2
Peer: 172.16.2.2+179 AS 65001  Local: 172.16.1.1+54265 AS 65001
  Group: EVPN-OVERLAY           Routing-Instance: master
  Forwarding routing-instance: master
  Type: Internal    State: Established    Flags: <Sync>
  Last State: OpenConfirm   Last Event: RecvKeepAlive
  Last Error: None
  Options: <Preference LocalAddress LogUpDown AddressFamily Rib-group Refresh>
  Address families configured: evpn
  Local Address: 172.16.1.1 Holdtime: 90 Preference: 170
```

For testing purposes we have provisioned two VMs in VLAN1500 connected on each QFX:

```
LAS-VMHOSTA  10.1.50.100   00:50:79:66:68:00
ORD-VMHOSTB 10.1.50.200   00:50:79:66:68:01
```

On the LASQFX-1 let's validate that the LAS-VMHOSTA MAC address is being learned locally:

```
admin@LASQFX-1> show ethernet-switching table | match 00:50:79:66:68:00
   VL1500              00:50:79:66:68:00   D        xe-0/0/1.0
```

Let's see if we are learning this MAC address across the overlay we just established. We should see Type 2 routes:

```
admin@ORDQFX-1> show route table bgp.evpn.0 | match 00:50:79:66:68:00
2:172.16.2.2:100::1500::00:50:79:66:68:00/304
2:172.16.2.2:100::1500::00:50:79:66:68:00::10.1.50.100/304
```

The ORDQFX-1 is now learning about this remote MAC address:

```
admin@ORDQFX-1> show ethernet-switching table | match 00:50:79:66:68:00
   VL1500              00:50:79:66:68:00   D        vtep.32769             172.16.2.2  .
```

On the ORDQFX-1 let's validate that the ORD-VMHOSTB MAC address is being learned locally:

```
admin@ORDQFX-1> show ethernet-switching table | match 00:50:79:66:68:01
   VL1500              00:50:79:66:68:01   D        xe-0/0/1.0
```

Let's see if we are learning this MAC address across the overlay just established. We should see Type 2 routes:

```
admin@LASQFX-1> show route table bgp.evpn.0 | match 00:50:79:66:68:01
2:172.16.1.1:1::1500::00:50:79:66:68:01/304
2:172.16.1.1:1::1500::00:50:79:66:68:01::10.1.50.200/304
```

The LASQFX-1 is now learning about this remote MAC address:

```
admin@LASQFX-1> show ethernet-switching table | match 00:50:79:66:68:01
   VL1500              00:50:79:66:68:01   D        vtep.32769             172.16.1.1
```

Now that we are learning the MAC addresses across the overlay let's validate that we have end-to-end connectivity:

```
ping from 10.1.50.100 to 10.1.50.200
LAS-VMHOSTA>ping 10.1.50.200
PING 10.1.50.200 (10.1.50.200): 56 data bytes
64 bytes from 10.1.50.200: icmp_seq=0 ttl=64 time=10.048 ms
64 bytes from 10.1.50.200: icmp_seq=1 ttl=64 time=9.079 ms
64 bytes from 10.1.50.200: icmp_seq=2 ttl=64 time=9.059 ms
64 bytes from 10.1.50.200: icmp_seq=3 ttl=64 time=9.060 ms
64 bytes from 10.1.50.200: icmp_seq=4 ttl=64 time=9.064 ms
```

```
2. ORD-VMHOSTB>ping 10.1.50.100
PING 10.1.50.200 (10.1.50.200): 56 data bytes
64 bytes from 10.1.50.100: icmp_seq=0 ttl=64 time=11.025 ms
64 bytes from 10.1.50.100: icmp_seq=1 ttl=64 time=10.072 ms
64 bytes from 10.1.50.100: icmp_seq=2 ttl=64 time=9.073 ms
64 bytes from 10.1.50.100: icmp_seq=3 ttl=64 time=9.050 ms
64 bytes from 10.1.50.100: icmp_seq=4 ttl=64 time=9.038 ms
```

Discussion

As with most networking solutions, there are always different ways to execute it or to modify the result. The original design requirement was to stretch a single VLAN across the overlay. With that as a requirement the QFX5100/QFX5200 can be utilized to provide this functionality.

Layer 3 was added for testing purposes. If any further routing, or inter-VXLAN routing is required that can be configured as well.

You could configure anycast gateways if inter-VXLAN routing or routing off the device is required – our example use case only requires stretching one VLAN for the purposes of VM migration.

Recipe 2: Identifying and Resolving Asymmetric Routing Problems

By Michel Tepper

- Junos OS Used: 18.1R2.5
- Juniper Platforms General Applicability: SRX, vSRX, MX

This recipe shows you how to recognize when asymmetric routing is occurring and offers various solutions for resolving it.

Problem

Asymmetric routing is probably as old as IP itself but before stateful firewalling it wasn't a problem. Nowadays, if a packet hits a stateful firewall with the initiating packet, but not with the return packet, the firewall will close the session. For TCP traffic this will kill the session.

Solution

First we'll describe how this can occur, and how to spot it when it's happening to you. Then we'll get to the solution – actually, several possible solutions.

When a SYN packet arrives at a SRX Series, the security device will follow the first path processing, and if routing is available to the destination, and a policy allows it, a session will be created with an initial timeout of 20 seconds. After the SRX detects that the three-way handshake has completed a SYN-ACK from the destination IP and an ACK from the source, the timeout is set to the timeout for the TCP application, 30 minutes by default.

So what if the SYN-ACK is not seen by the SRX? Then the session is closed after 20 seconds, with the reason cited as *age-out*.

An example of this situation could be: We have a "normal" network with a internal network, a SRX and a Internet connection:

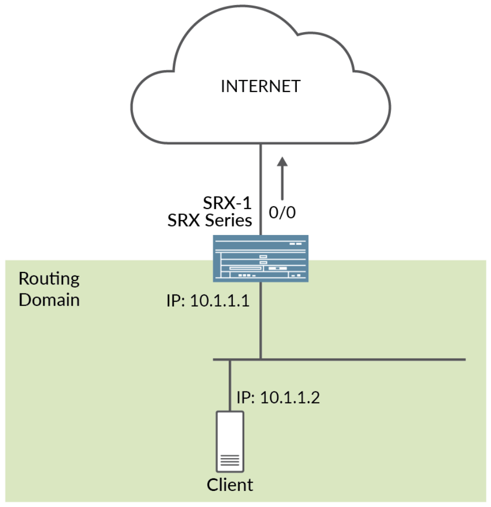


Figure 2.1 Normal Network

Now a private cloud service that's reachable over a leased line is added to this network as shown in Figure 2.2. The cloud provider places a SRX (SRX-2) on premises and is given the IP address 10.1.1.3 as local gateway IP. The server in the cloud runs on IP 10.1.2.2.

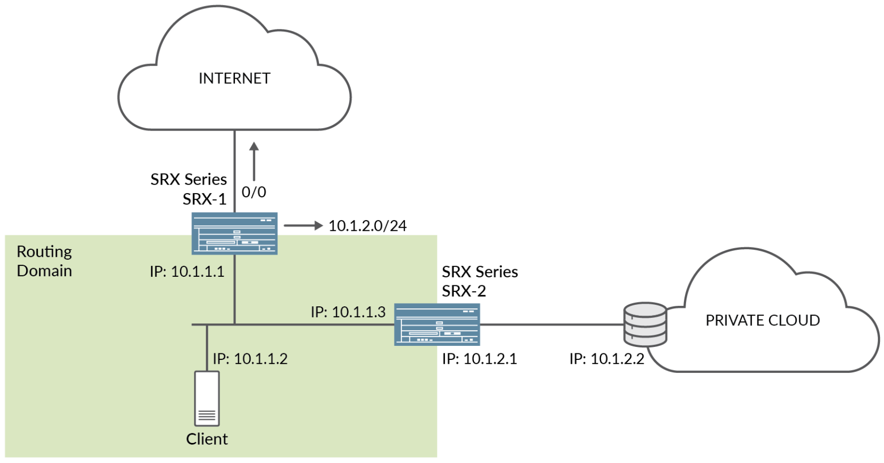


Figure 2.2 Normal Network with Private Cloud

The local admin knows how to add a static route on the SRX Services and configures this:

```
set routing-options static route 10.1.2.0/24 next-hop 10.1.1.3
```

The routing table on SRX 1 now holds two routes:

```
root@SRX-1# run show route

inet.0: 8 destinations, 8 routes (8 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

0.0.0.0/0          *[Static/5] 00:00:17
                    > to 1.1.3.2 via ge-0/0/0.0

.......................................

10.1.2.0/24        *[Static/5] 00:02:18
                    > to 10.1.1.3 via irb.0
.......................................
```

Nothing has changed on the client, so the client is sending traffic for 10.1.2/24 to its default gateway SRX-1on 10.1.1.1. SRX-1 has a static route for this prefix to SRX-2, and the packet arrives at the destination as shown in Figure 2.3. The server sends its reply to SRX-2. SRX-2 sees that the source IP address, 10.1.1.2, is the client. It has a directly connected route to this client, so it does an ARP request and sends the packet directly to the client, ignoring SRX-1. SRX-1 has seen the SYN, but not the SYN-ACK.

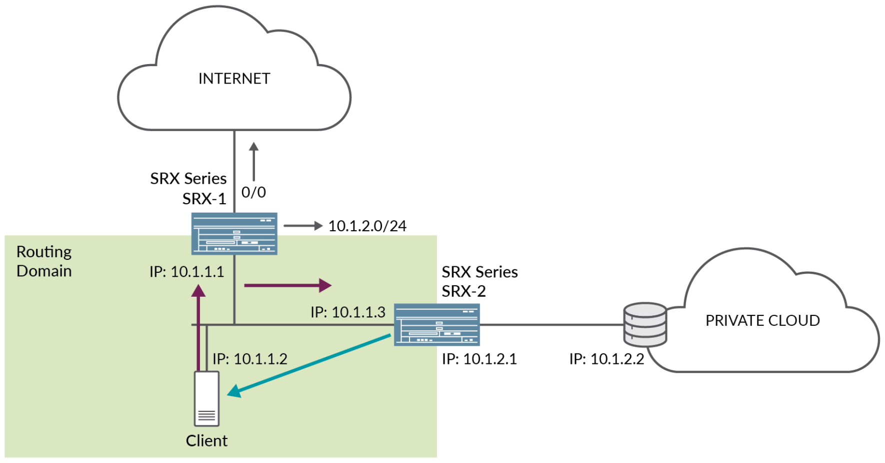


Figure 2.3 *Normal Network with Asymmetry*

Let's see what happens with a ping from client to server:

```
root@CLIENT> ping 10.1.2.2
PING 10.1.2.2 (10.1.2.2): 56 data bytes
64 bytes from 10.1.2.2: icmp_seq=0 ttl=62 time=25.771 ms
64 bytes from 10.1.2.2: icmp_seq=1 ttl=63 time=1.800 ms
```

```
64 bytes from 10.1.2.2: icmp_seq=2 ttl=63 time=1.814 ms
64 bytes from 10.1.2.2: icmp_seq=3 ttl=63 time=1.919 ms
^C
```

Good, the line seems to work and the server is responding! Let's set up a SSH session to this server:

```
root@CLIENT> ssh 10.1.2.2
Password:
Last login: Tue May 29 17:33:54 2018 from 10.31.5.66
--- JUNOS 18.1R2.5 built 2018-05-25 20:42:45 UTC
root@EX:RE:0%
root@EX:RE:0% packet_write_wait: Connection to 10.1.2.2 port 22: Broken pipe
```

That's strange: it seems we can log in, but quickly after we do the session becomes non-responsive! After that we get a broken pipe message.

"Quickly" in this case means around 20 seconds after the connection is established, so about 15 seconds after the login.

The ping got its reply (directly from SRX-2), but the TCP session was cancelled after the SRX didn't see a SYN-ACK after 20 seconds. For ICMP (and UDP) you won't notice a problem because both are stateless and the client and the server are able to reach each other. In the security log you will see packets from client to server, but no return packets and the reason will be a *timeout*.

Recognizing the Problem

The easiest way to find asymmetric routing in an open environment is to do a traceroute from client to server, and another one from server to client. The same hops should be seen in reverse order. Unfortunately the biggest problems with asymmetric routing arise in environments where SRXs (or other stateful firewalls) are used. And because of NAT rules or policies traceroute just might not work. So this is not reliable enough.

Let's take a look at the traffic log on SRX-1 from this recipe's reference topology and try to carefully examine what we can see. Of course, you have to make sure traffic is logged.

NOTE Enabling logging may sound obvious, but traffic isn't logged by default. Be careful not to enable event-based logging on all traffic when you're not sure of the impact of logging. To enable logging on the box:

```
set security log mode event
```

Add `then log session-close` to the policy you want logging on. If that policy might be hit by too much traffic just place a more specific one before it.

All that's left to enable the logging now is this:

```
set system syslog file traffic-log any any
set system syslog file traffic-log match "RT_FLOW_SESSION”
```

We’re looking for a ping from 10.1.1.2 to 10.1.2.2. It’s a good idea to use the available tools on the CLI to help here. In the logging, a `source-ip` is logged first, then a `destination-ip` in the same line. So matching with the regular expression `<source-ip>.*<destination-IP>` should show all log entries for traffic from this source to this destination:

```
root@SRX-1> show log policy_session | match "10.1.1.2.*10.1.2.2"
<14>1 2018-06-12T13:12:21.082Z SRX-1 RT_FLOW - RT_FLOW_SESSION_
CLOSE [junos@2636.1.1.1.2.135 reason="idle Timeout" source-address="10.1.1.2" source-
port="0" destination-address="10.1.2.2" destination-port="4642" connection-tag="0" service-
name="icmp" nat-source-address="10.1.1.2" nat-source-port="0" nat-destination-
address="10.1.2.2" nat-destination-port="4642" nat-connection-tag="0" src-nat-rule-type="N/A" src-
nat-rule-name="N/A" dst-nat-rule-type="N/A" dst-nat-rule-name="N/A" protocol-id="1" policy-
name="trust-to-trust" source-zone-name="trust" destination-zone-name="trust" session-
id-32="8" packets-from-client="1" bytes-from-client="84" packets-from-server="0" bytes-from-
server="0" elapsed-time="59" application="UNKNOWN" nested-
application="UNKNOWN" username="N/A" roles="N/A" packet-incoming-
interface="irb.0" encrypted="UNKNOWN" application-category="N/A" application-sub-
category="N/A" application-risk="-1"] session closed idle Timeout: 10.1.1.2/0-
>10.1.2.2/4642 0x0 icmp 10.1.1.2/0->10.1.2.2/4642 0x0 N/A N/A N/A N/A 1 trust-to-
trust trust trust 8 1(84) 0(0) 59 UNKNOWN UNKNOWN N/A(N/A) irb.0 UNKNOWN N/A N/A -1
```

What’s in this structured formatted log line?:

- It’s a session closed log: `RT_FLOW_SESSION_CLOSE`
- The reason for closing the session is a time out: `reason="idle Timeout"`
- The source IP: `source-address="10.1.1.2"`
- The destination IP: `destination-address="10.1.2.2"`
- The service of this packet: `service-name="icmp"`
- The amount of data from client to server: `packets-from-client="1" bytes-from-client="84"`
- The amount of data from server to client by SRX-1: `packets-from-server="0" bytes-from-server="0”`

To summarize: it’s a successful ping but the SRX only saw the initial packet, not the return packet. You know this because the close reason would otherwise have been *response received*, and you would have seen return traffic from the server to the client. This alone should make you suspect asymmetric routing. To confirm let’s look at the SSH log line:

```
root@SRX-1>show log policy_session |match "10.1.1.2.*10.1.2.2.*ssh"
<14>1 2018-06-12T14:06:23.084Z SRX-1 RT_FLOW - RT_FLOW_SESSION_
CLOSE [junos@2636.1.1.1.2.135 reason="idle Timeout" source-address="10.1.1.2" source-
port="60328" destination-address="10.1.2.2" destination-port="22" connection-tag="0" service-
name="junos-ssh" nat-source-address="10.1.1.2" nat-source-port="60328" nat-destination-
address="10.1.2.2" nat-destination-port="22" nat-connection-tag="0" src-nat-rule-type="N/A" src-nat-
rule-name="N/A" dst-nat-rule-type="N/A" dst-nat-rule-name="N/A" protocol-id="6" policy-name="trust-
to-trust" source-zone-name="trust" destination-zone-name="trust" session-id-32="62" packets-from-
client="60" bytes-from-client="5445" packets-from-server="0" bytes-from-server="0" elapsed-
time="19" application="UNKNOWN" nested-application="UNKNOWN" username="N/A" roles="N/A" packet-
incoming-interface="irb.0" encrypted="UNKNOWN" application-category="N/A" application-sub-
category="N/A" application-risk="-1"] session closed idle Timeout: 10.1.1.2/60328-
>10.1.2.2/22 0x0 junos-ssh 10.1.1.2/60328->10.1.2.2/22 0x0 N/A N/A N/A N/A 6 trust-to-
trust trust trust 62 60(5445) 0(0) 19 UNKNOWN UNKNOWN N/A(N/A) irb.0 UNKNOWN N/A N/A -1
```

This log shows a session, with a reason of *idle timeout*, a duration of 19 seconds (`elapsed-time="19"`), 60 packets from client to server, but no return packets, and not one returned byte, not even the SYN-ACK. The SRX is being bypassed in the return flow. There must be asymmetric routing going on here.

Implementing the Solution

Let's move to the solution. There are some very good solutions, some very bad solutions, and some in between. Some of the possible good solutions are:

- Redesign the routing. Connect SRX-2 to a free port on SRX-1 and use a /30 network in between. This is probably the cleanest solution.
- Fix it on the client. Create a routing entry on the client instructing it to use 10.1.1.3 as the gateway to reach 10.1.2/24.
- Fix it on SRX-2 by setting a /32 route to reach 10.1.1.2 to gateway 10.1.1.1: `set routing-options static route 10.1.1.2 next-hop 10.1.1.1`. This will work for a few clients but it is not scalable.
- Fix it using `source-nat` on SRX-1. If you NAT the traffic on SRX-1 to the outgoing IP address, the server sees SRX-1 as the source. Return traffic is sent to SRX-1 and then forwarded to the client. The routing is solved, but the logging on the server is ruined. Also, traffic coming in from the client network gets 10.1.1.1 as its source IP in the logging. An example for the NAT configuration is here:

```
root@SRX-1# show security nat source rule-set trust-to-trust
from zone trust;
to zone trust;
rule intrazone-trust {
    match {
        source-address 0.0.0.0/0;
    }
    then {
        source-nat {
            interface;
        }
    }
}
```

NOTE The ugliest solution is to instruct SRX-1 to forget about SYN checking at all (`set security flow tcp-session no-syn-check`). It will work but it is very insecure. Never tell your security officer if you opt for this one.

You might consider disabling `syn-check` globally and use `apply-groups` on the policy hierarchy to enable again on a per policy basis. On the one policy you don't want syn-checking on you then use **`apply-group-accept`** on that particular policy's level. It works, but stays an ugly solution!

Discussion

Recognizing asymmetric routing might still be a little difficult. It takes some practice to interpret the traffic log correctly. But remember that whenever you have TCP connections failing after about 20 seconds asymmetric routing is most likely your primary suspect!

Choosing the right solution depends on circumstances and personal preferences. In general, redesigning the network is always the best solution, with NAT as a second. Just look at the pros and cons from every angle.

Recipe 3: Configuring Filter-Based Forwarding Inside a Junos Routing Instance

By Steve Puluka

- Junos OS Used: 13.3
- Juniper Platforms General Applicability: SRX; MX; EX; QFX

The Junos OS uses the technique of filter-based forwarding (FBF) to override the normal, destination-based routing used by all protocols. These techniques allow routing to occur based on other features of the traffic such as source address, or protocol and ports, or combinations of destination, source, and protocol and port.

Problem

All of Juniper's Junos documentation shows how to configure FBF based on the forwarding action occurring in the main, or base, Junos routing instance. But how do you implement FBF when the traffic being processed arrives at an interface *inside* of a virtual router's (VR) routing instance?

Solution

You can configure FBF within a Junos routing instance by following a few configuration steps. First, let's look at the example network topology and forwarding problem in Figure 3.1.

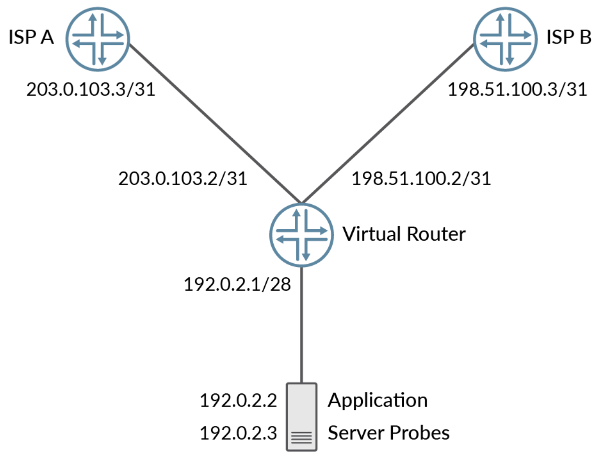


Figure 3.1 *Network Diagram Monitoring Application Server with Dual ISP*

You can see Figure 3.1 has two ISP connections, A and B, that terminate in a Junos VR routing instance. The monitoring application has two IP addresses assigned and uses a different IP address to send probes out to each of the connected ISP links. This allows the same destination address to be probed and tested for performance from the server overriding the single routing choice for that single destination currently installed in the shared VR routing instance. So, in this use case, the match criteria for the FBF function will be the source address coming from the probe server, one for each ISP.

The configuration needed for the forwarding process contains the following elements:

First is the filter. The match criteria filter is used to select which traffic is affected. You create a single filter that has the two match conditions, one for the IP address associated with each ISP. Then there is a final term that passes all other traffic unaffected:

```
set firewall family inet filter isp_fbf term ispa from source-address 192.0.2.2/32
set firewall family inet filter isp_fbf term ispa then routing-instance ispa
set firewall family inet filter isp_fbf term ispb from source-address 192.0.2.3/32
set firewall family inet filter isp_fbf term ispb then routing-instance ispb
set firewall family inet filter isp_fbf term final then accept
```

The second part is the interface. Here the filter is applied to the interface where the traffic enters the VR routing instance:

set interfaces ge-0/0/0 unit 0 family inet filter input isp_fbf

Third are the base Junos routing options. This creates a `rib-group` for the route leaking with the forwarding routing instances that are referenced in the forwarding routing instance:

```
set routing-options rib-groups isp_fbf import-rib upstream.inet.0
set routing-options rib-groups isp_fbf import-rib ispa.inet.0
set routing-options rib-groups isp_fbf import-rib ispb.inet.0
```

Next is the forwarding routing instance. There is a forwarding routing instance for each ISP that you want to use FBF for forwarding. This contains the route forcing traffic to the respective ISP:

```
set routing-instances ispa instance-type forwarding
set routing-instances ispa routing-options static route 0.0.0.0/0 next-hop 203.0.103.3
set routing-instances ispb instance-type forwarding
set routing-instances ispb routing-options static route 0.0.0.0/0 next-hop 198.51.100.3
```

Fifth is the VR routing instance `rib-group`. In the existing upstream VR routing instance, you add the `rib-group` created in the base routing options to leak the routes from our forwarding routing instances. This also shows that the three interfaces are all assigned to the upstream VR routing instance:

```
set routing-instances upstream instance-type virtual-router
set routing-instances upstream interface ge-0/0/0.0
set routing-instances upstream interface ge-0/0/1.0
set routing-instances upstream interface ge-0/0/2.0
set routing-instances upstream routing-options interface-routes rib-group inet isp_fbf
```

Now for verification. Check the routing tables to confirm the new forwarding instance's default routes are present. The forwarding instance's routing tables should contain all the interface routes from the upstream VR plus the default route that was installed in the routing table. This insures that when the filter forwards the matching packets to this forwarding instance, they will be sent to the desired ISP:

```
root@upstream-1> show route table ispa

ispa.inet.0: 7 destinations, 7 routes (7 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

0.0.0.0/0          *[Static/5] 00:00:27
                    > to 203.0.103.3 via ge-0/0/1.0
192.0.2.1/31      *[Direct/0] 00:00:41
                    > via ge-0/0/0.0
192.0.2.1/32      *[Local/0] 00:00:41
                      Local via ge-0/0/0.0
203.0.103.2/31     *[Direct/0] 00:00:41
                    > via ge-0/0/1.0
203.0.103.2/32     *[Local/0] 00:00:41
                      Local via ge-0/0/1.0
198.51.100.2/31     *[Direct/0] 00:00:41
                    > via ge-0/0/2.0
198.51.100.2/32     *[Local/0] 00:00:41
                      Local via ge-0/0/2.0

root@upstream-1> show route table ispb

ispa.inet.0: 7 destinations, 7 routes (7 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both
```

```
0.0.0.0/0          *[Static/5] 00:00:27
                    > to 198.51.100.3 via ge-0/0/1.0
192.0.2.1/31       *[Direct/0] 00:00:41
                    > via ge-0/0/0.0
192.0.2.1/32       *[Local/0] 00:00:41
                      Local via ge-0/0/0.0
203.0.103.2/31     *[Direct/0] 00:00:41
                    > via ge-0/0/1.0
203.0.103.2/32     *[Local/0] 00:00:41
                      Local via ge-0/0/1.0
198.51.100.2/31     *[Direct/0] 00:00:41
                    > via ge-0/0/2.0
198.51.100.2/32     *[Local/0] 00:00:41
                      Local via ge-0/0/2.0
```

Discussion

This example used matching conditions of the specific source address used by the host traffic. Its application is making sure monitoring probes go down specified paths regardless of the installed routing table. Network facilities like Noction Route Optimizers use this type of function.

The same source address matching can also be used to allocate ISP by LAN subnet. Instead of using the /32 match, you can match on the /24 allocated to various VLANs and assign different groups of users to particular ISP connections. You can also reference prefix lists in the filter so that subnets can be added and removed without changing the filter itself.

The same technique can also be used for other match conditions in the filter, for example, to use make sure ISP A has all web browsing traffic while ISP B is the source for SMTP traffic outbound. For these types of situations you modify the filter to match on protocols instead of source address:

```
set firewall family inet filter isp_fbf term ispa from destination-port http
set firewall family inet filter isp_fbf term ispa from destination-port https
set firewall family inet filter isp_fbf term ispa then routing-instance ispa
set firewall family inet filter isp_fbf term ispb from destination-port smtp
set firewall family inet filter isp_fbf term ispb then routing-instance ispb
set firewall family inet filter isp_fbf term final then accept
```

In a similar way, many criteria for the forwarding of the traffic are available. Using DSCP markings allows you to manipulate via class of service filters, and a variety of other criteria are possible. Using the `?` in the filter as you configure will show you the many options you can match on:

```
edit firewall family inet filter isp_fbf
Set term ispa from ?
Possible completions:
> address              Match IP source or destination address
+ apply-groups         Groups from which to inherit configuration data
+ apply-groups-except  Don't inherit configuration data from these groups
> destination-address  Match IP destination address
```

```
+ destination-port     Match TCP/UDP destination port
+ destination-port-except  Do not match TCP/UDP destination port
> destination-prefix-list  Match IP destination prefixes in named list
+ dscp                 Match Differentiated Services (DiffServ) code point
+ dscp-except          Do not match Differentiated Services (DiffServ) code point
+ esp-spi              Match IPSec ESP SPI value
+ esp-spi-except       Do not match IPSec ESP SPI value
  first-fragment       Match if packet is the first fragment
+ forwarding-class     Match forwarding class
+ forwarding-class-except  Do not match forwarding class
  fragment-flags       Match fragment flags (in symbolic or hex formats) - (Ingress only)
+ fragment-offset      Match fragment offset
+ fragment-offset-except  Do not match fragment offset
+ icmp-code            Match ICMP message code
+ icmp-code-except     Do not match ICMP message code
+ icmp-type            Match ICMP message type
+ icmp-type-except     Do not match ICMP message type
> interface            Match interface name
+ interface-group      Match interface group
+ interface-group-except  Do not match interface group
> interface-set        Match interface in set
+ ip-options           Match IP options
+ ip-options-except    Do not match IP options
  is-fragment          Match if packet is a fragment
+ packet-length        Match packet length
+ packet-length-except  Do not match packet length
+ port                 Match TCP/UDP source or destination port
+ port-except          Do not match TCP/UDP source or destination port
+ precedence           Match IP precedence value
+ precedence-except    Do not match IP precedence value
> prefix-list          Match IP source or destination prefixes in named list
+ protocol             Match IP protocol type
+ protocol-except      Do not match IP protocol type
  service-filter-hit   Match if service-filter-hit is set
> source-address       Match IP source address
+ source-port          Match TCP/UDP source port
+ source-port-except   Do not match TCP/UDP source port
> source-prefix-list   Match IP source prefixes in named list
  tcp-established      Match packet of an established TCP connection
  tcp-flags            Match TCP flags (in symbolic or hex formats)
  tcp-initial          Match initial packet of a TCP connection
+ ttl                  Match IP ttl type
+ ttl-except           Do not match IP ttl type
```

Recipe 4: Configuring DHCP Services on the EX2300/ EX4300 Series with Enhanced Layer 2 Software

By Jeffrey Fry

- EX4200 Version Used: 12.3R12.4
- EX4300 Version Used: 14.1X53-D46.7

This recipe will help you convert your legacy EX4200/EX2200 DHCP configuration over to the new *Enhanced Layer 2 Software* (ELS).

Problem

Traditionally the EX2200 and EX4200 Series were the go-to switches for an enterprise or small business. Businesses ran these as their core switches, and commonly had DHCP configured on them to serve IP addresses to the end workstations.

Now businesses are replacing their EX2200s and EX4200s with the next generation of switches, the EX2300 and EX4300. With this new hardware comes new versions of Junos that support a new syntax called ELS. One of the services that has changed in ELS is DHCP, meaning that the legacy DHCP configuration is no longer compatible with the ELS version of Junos.

The new DHCP process, called JDHCP, brings the DHCP configuration of ELS-based switches in line with other Juniper products such as the MX. This new process also brings advanced features such as easier configuration of DHCP services in virtual routing instances, and a single configuration point for both DHCPv4 and DHCPv6 as a few examples.

This recipe will allow you to convert your legacy DHCP configuration the new ELS syntax. Let's get started!

Solution

To address this problem you must convert your existing EX2200/EX4300 Series configuration over to an ELS configuration. This recipe will cover the common configuration elements to help get your new switches to act as a DHCP server or forward the DHCP packets to a server.

In the first example, let's use the switch as a DHCP server and take the EX4200 DHCP server configuration and convert it to an EX4300 configuration.

Figure 4.1 depicts what our network looks like for this lab. There's a client on each switch so that we can see active leases.

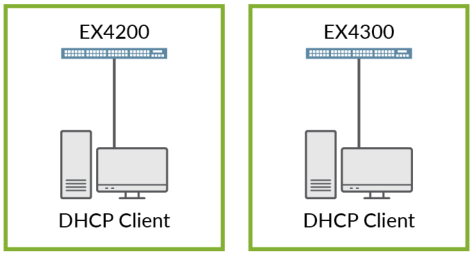


Figure 4.1 An EX Series Acting as DHCP Server

Here is the DHCP configuration for an EX4200 that is acting as the DHCP server. You will convert the configuration to show what it looks like on an EX4300.The DHCP pool is configured with the subnet of 192.0.2.0/24, all the ports are assigned to a VLAN called `WIRED`, and there is an IP address of 192.0.2.1/24 on the vlan.100 interface:

```
system {
    services {
            pool 192.0.2.0/24 {
                address-range low 192.0.2.25 high 192.0.2.250;
                name-server {
                    192.0.2.5;
                    192.0.2.6;
                }
                router {
                    192.0.2.1;
                }
            }
        }
    }
}
interfaces {
    interface-range ACCESS-PORTS {
        member-range ge-0/0/0 to ge-0/0/47;
        unit 0 {
            family ethernet-switching {
                vlan {
                    members WIRED;
                }
            }
        }
    }
    vlan {
        unit 100 {
            family inet {
                address 192.0.2.1/24;
            }
        }
    }
}
vlans {
    WIRED {
        vlan-id 100;
        l3-interface vlan.100;
    }
}
```

The first thing that is different in ELS software is that DHCP is enabled on the `irb interface` under `system services dhcp-local-server` stanza. DHCP features, such as the address pool, any options such as DNS/Gateway/TFTP servers are configured under the `access` stanza. The configuration to enable a local DHCP server on the `WIRED` VLAN is shown below.

IMPORTANT One of the major differences between the legacy and ELS software is that the L3 VLAN interfaces has changed from `vlan` to `irb`. While ELS will allow you to configure and commit a `vlan` interface configuration, the VLAN interfaces will not work.

To help simplify your configuration you should configure the DHCP group name to match the name of your configured VLAN. From there add the `irb` interface to the group under the `system services dhcp-local-server group WIRED` section.

The `pool-match-order` is used to determine where to pull the DHCP IP address from. You are able to configure `ip-address-first`, `external-authority`, and `option-82`. The example here is `ip-address-first` which uses the gateway IP in the DHCP client PDU:

```
system {
    services {
        dhcp-local-server {
            pool-match-order {
                ip-address-first;
            }
            group WIRED {
                interface irb.100;
            }
        }
    }
}
```

```
interfaces {
    interface-range ACCESS-PORTS {
        member-range ge-0/0/0 to ge-0/0/47;
        unit 0 {
            family ethernet-switching {
              interface-mode access;
                vlan {
                    members WIRED;
                }
            }
        }
    }
    irb {
        unit 100 {
            family inet {
                address 192.0.2.1/24;
            }
        }
    }
}
vlans {
    WIRED {
        vlan-id 100;
        l3-interface irb.100;
    }
}
```

The access configuration is similar to what we had before, but instead of DHCP, you place it under `address-assignment`. Define your network, range, and any additional DHCP attributes:

```
access {
    address-assignment {
        pool WIRED {
            family inet {
                network 192.0.2.0/24;
                range WIRED_CLIENTS {
                    low 192.0.2.25;
                    high 192.0.2.250;
                }
                dhcp-attributes {
                    name-server {
                        192.0.2.5;
                        192.0.2.6;
                    }
                    router {
                        192.0.2.1;
                    }
                }
            }
        }
    }
}
```

To check to see if your switch is working and acting as a DHCP server, issue the `show dhcp server statistics` command. This is the same command that we used on non-ELS Junos:

```
user1@lab> show dhcp server statistics
Packets dropped:
    Total                      0

Messages received:
    BOOTREQUEST                1
    DHCPDECLINE                0
    DHCPDISCOVER               0
    DHCPINFORM                 1
    DHCPRELEASE                0
    DHCPREQUEST                0

Messages sent:
    BOOTREPLY                  1
    DHCPOFFER                  0
    DHCPACK                    1
    DHCPNAK                    0
    DHCPFORCERENEW             0
```

Everything is up and running.

Your next step is to configure the EX4300 as a DHCP relay. Take a look at Figure 4.2 – the DHCP service is running on an external server instead of locally on the switch. This means that the EX4300 will be forwarding DHCP requests to the DHCP Server.

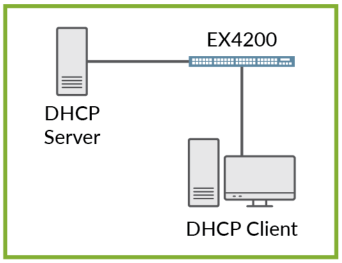


Figure 4.2 *EX Series Forwards DHCP to an External Server*

In the EX4200/2200 Series switches the DHCP Relay feature is configured under `forwarding-options helpers` stanza as shown below:

```
forwarding-options {
    helpers {
        bootp {
            server 192.0.2.9;
            interface {
                vlan.100;
            }
        }
    }
}
```

In ELS the DHCP Relay feature is configured under `forwarding-options server-group` and the `forwarding-options group` stanzas.

You will need to create a `DHCP_SERVER` group that lists the IP address of one or more DHCP servers. From there you can configure the group by setting the previously configured server-group as the `active-server-group`, and assign one or more IRB interfaces to the configuration:

```
dhcp-relay {
    server-group {
        DHCP_SERVER {
            192.0.2.9;
        }
    }
    group DHCP-SERVER {
        active-server-group DHCP_SERVER;
        interface irb.100;
    }
```

Once that is committed your switch will be forwarding DHCP requests to the DHCP server. You can check to see if it is working by issuing the `show dhcp relay statistics` command:

```
user1@lab> show dhcp relay statistics
Packets dropped:
    Total                      0
    dhcp-service total         0

Messages received:
    BOOTREQUEST                1
    DHCPDECLINE                0
    DHCPDISCOVER               0
    DHCPINFORM                 1
    DHCPRELEASE                0
    DHCPREQUEST                0

Messages sent:
    BOOTREPLY                  1
    DHCPOFFER                  0
    DHCPACK                    1
    DHCPNAK                    0
    DHCPFORCERENEW             0
```

Some network devices such as printers need to lease the same IP Address consistently. You can accomplish this by creating a DHCP reservation. Using our example above you can create this reservation under the `dhcp-attributes` using `host`. In this next example, a host `PRINTER` with a MAC address of 00:00:5E:00:53:00 is assigned an IP address of 192.0.2.15:

```
access {
    address-assignment {
        pool WIRED {
            family inet {
                network 192.0.2.0/24;
                dhcp-attributes {
                    host PRINTER {
                    hardware-address 00:00:5E:00:53:00;
                    ip-address 192.0.2.15;
                    }
                }
            }
        }
    }
}
```

Once you have that configured and the client comes online, you can see the lease by using the `show dhcp server binding 192.0.2.15` command:

```
user1@lab> show dhcp server binding 192.0.2.15

IP address        Session Id  Hardware address   Expires     State      Interface
192.0.2.15        4           00:00:5E:00:53:00  13114       BOUND      irb.100
```

Some devices use DHCP Option to configure services such as a where to find a wireless controller. A common use case is DHCP Option 43, which is used by Wireless Access Points (WAPs) to find a centralized wireless controller. DHCP options are configured for an entire DHCP pool under the `dhcp-attributes` stanza. The example below shows how to configure DHCP Option 43 using the 192.0.2.10 as the wireless controller's IP address:

```
access {
    address-assignment {
        pool WIRED {
            family inet {
                network 192.0.2.0/24;
                dhcp-attributes {
                    option 43 ip-address 192.0.2.10;
                    }
                }
            }
        }
    }
}
```

Discussion

You can find more information about `pool-match-order` and its options in the Juniper TechLibrary: https://www.juniper.net/documentation/en_US/junos/topics/reference/configuration-statement/pool-match-order-edit-system-services.html.

Security is critical in any infrastructure, understanding DHCP Snooping is something that will help to prevent rogue DHCP servers. You can read more about that here: https://www.juniper.net/documentation/en_US/junos/topics/concept/port-security-dhcp-snooping-els.html.

Recipe 5: Exceeding the MTU and GRE Tunnels

By Martin Brown

- Junos OS Used: 12.3X48-D40.5
- Juniper Platforms General Applicability: SRX Series

Without the niceties of QoS, modern networks attempt to make bandwidth allocation fair for all network users, and restricting the size of the packet being sent is one of the methods employed to ensure all users can send data on the network when they need to. While this method solves a few issues, it can also create issues with other network services such as GRE tunnels. This recipe looks at how restricting packet sizes can affect GRE tunnels and how you can overcome these restrictions so that they don't bring your tunnel down.

Problem

If a router is about to send a packet, and that packet exceeds the MTU set on an interface, the router has only two choices: drop the packet or fragment the packet.

Solution

A number of years ago, when Ethernet and Internet Protocol were first being developed, designers struggled to decide what the maximum size of packets and frames should be. While they were deciding on this, another decision was made to add a field to the packet header which described to the receiving device how big the packet was, this in turn allowed the receiver to check if anything was missing and how much more of the packet to expect.

The size of this field is two bytes in length, which, if you know binary, means the value can be anything between 0 and 65535. As this field is the size in bytes, it means the maximum theoretical packet size is up to 65535 bytes or 63KB. In today's modern networks, 63KB isn't a huge deal. A 1GB network has a throughput of around 118MB per second, but back when IP was being developed, some corporate networks were 4Mb/s for Token Ring, or 10Mb/s for 10Base2, and these networks were shared media, meaning only one host could send data at any one time.

Imagine, then, as in Figure 5.1, that you have two clients connected to a shared media network. Both of these clients wish to send a packet to a database server on another subnet.

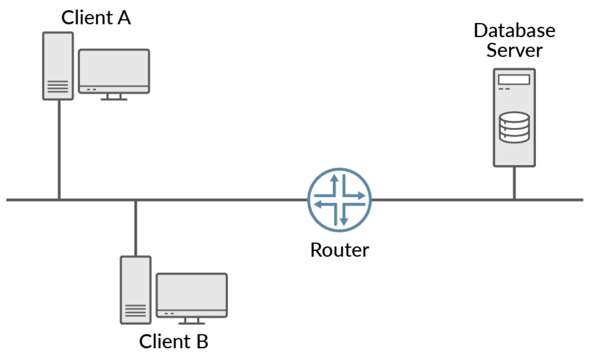


Figure 5.1 *Two Clients On a 10Base2 Network and a Database Server*

Now imagine that both clients wish to send data to the database server at the same time. Client A wants to send a packet that is only 100 bytes in size, however, Client B wishes to send a packet that is 65535 bytes in size, and Client B got there first. Client A will now need to wait an unreasonable amount of time for Client B to finish sending his frame to the router, but will need to wait for the router to then process and forward the packet as well.

As you can see, having a packet that is too large on a slow network could have an adverse effect on other network users, however, it wasn't just the network speed that was a factor in deciding how large a packet should be. At the time memory was expensive, and a 1MB SIMM was, weight for weight, worth more than gold (and when a router receives a packet, it must first place this in a buffer before processing). Because a buffer is basically memory, the smaller you can make the buffer the cheaper the router will be, not only to manufacture, but for the end user to purchase.

Recognizing that having packets as large as 63KB wasn't ideal, the designers introduced a mechanism to restrict packet sizes. This method became known as maximum transmission unit (MTU). Basically, packet sizes were restricted to whatever the MTU was set to and the default was usually set to 1500 bytes.

NOTE Having the MTU set to 1500 bytes is not a hard and fast rule. Network administrators can, and do, change MTUs when needed, and it is possible to find an MTU on one interface on one router in a network path set to less than 1500 bytes.

Normally, having the MTU set as default isn't a huge issue. Most applications and routers expect the MTU to be that size and will send packets accordingly. Most applications and routers, however, do not expect a device or service in the network path to add an additional overhead to the MTU and unfortunately, this can and does happen, especially when that service is a GRE tunnel.

GRE tunnels are great; they allow network traffic to be sent inside another IP packet. The possibilities when using GRE tunnels are almost endless, however the common uses for GRE tunnels include:

- Obfuscation of network traffic
- Sending data with private IP addresses across the public Internet
- Sending non-IP traffic such as AppleTalk over an IP network
- Adding cloud-based proxy services to a corporate network
- Connecting branch offices to HQs

GRE tunnels don't perform their magic by actually digging a tunnel through the Internet; first, where would you put the bits of cloud you just dug up, and second, it would take too long to actually dig the tunnel. No, in fact, GRE tunnels work by encapsulating IP packets inside another IP packet, or in the case of IPv6 packets and protocol data units (PDUs), just inside a single IP packet. In real terms, this basically means putting an additional header in front of the existing packet or PDU.

Imagine, therefore, that your client in a branch has just sent a packet to a server located in the HQ. Your company is connecting the branch to the HQ using a GRE tunnel, which is traversing the public Internet, as shown in Figure 5.2.

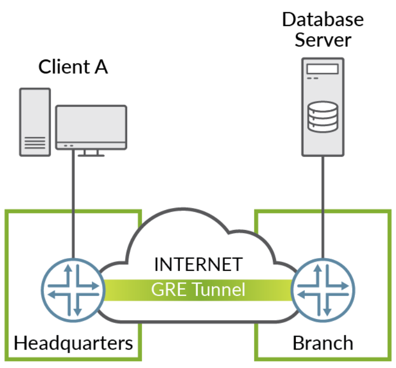


Figure 5.2 Branch and HQ Connected via a GRE Tunnel

The client in this situation is sending a packet, which is 1500 bytes in size. Under normal circumstances this wouldn't be a problem, but because the IP packet from the client is already equal to the MTU, when the packet is encapsulated inside another IP packet the MTU is then exceeded (though not by much, as you can see in Figure 5.3).

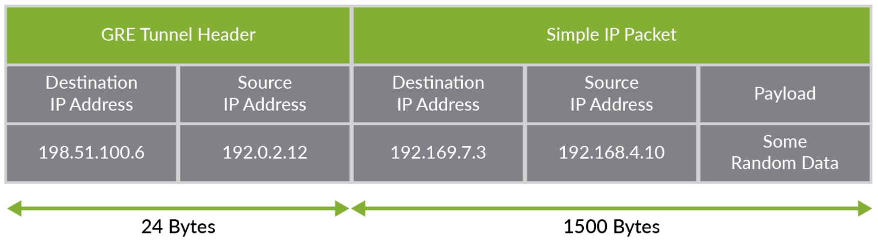


Figure 5.3 *GRE Encapsulated IP Packet*

The GRE tunnel header is basically an IP header without the data, which is 20 bytes in length, and an additional GRE header, which is 4 bytes in length. The additional 24 bytes will increase the packet size to 1524 bytes. If you were buying groceries from a shop and the goods came to $15.24 and you only had $15, the cashier might say, "It's okay, I'll let you off the 24 cents." Routers, however, are not so generous. They have no leniency and they are almost always black and white. If a packet exceeds the MTU set on an interface, the router has only one of two choices:

- Drop the packet

or

- Fragment the packet

If a packet is a part of a GRE tunnel and the packet is fragmented, then this isn't too much of an issue, the packet will be delivered. If, however, the packet is dropped, then the tunnel may go down.

Routers will only fragment the packet if they are allowed to do so, and the way you can tell is if the DF, or "Do Not Fragment," bit in the header is turned off. If this bit is turned on, then the router has no choice but to drop the packet, meaning your tunnel will go down.

For some reason, when routers encapsulate a packet within a GRE header, they have a habit of automatically turning the DF bit on, meaning the router will itself cause a packet that the router knows will exceed the MTU to be dropped. This is unacceptable.

Happily, within the Junos OS, there is a solution.

Imagine that we are configuring a GRE tunnel between ACME's HQ and branch office. In the HQ and branch office are SRX Series firewalls connected directly to the Internet, with a static default route going to the ISP. At the moment, ACME is not bothered about encryption, they just want the basic tunnel configured, so let's configure these tunnels now.

Similar to most interfaces in Junos OS, GRE tunnel interfaces always start with 'gr' followed by numbers which represent the FPC the interface is on, the PIC the interface is on, and the port number itself. GRE tunnel interfaces always have a sub interface number, too. In ACME's case, the HQ's interface will be named gr-0/0/0 and it will have a sub interface of 10:

```
edit interface gr-0/0/0.10
```

The tunnel source is the HQ firewall's external interface address, which is 203.0.113.16 and the destination is the IP address of the branch SRX's external interface, which is 192.0.2.10:

```
set tunnel source 203.0.113.16
set tunnel destination 192.0.2.10
```

Finally, in order to allow the interfaces to become part of an OSPF domain, the tunnel interface is given an IP address of 192.168.1.1/30:

```
set family inet address 192.168.1.1/30
```

Once this is done, you need to perform the opposite on the branch SRX with the tunnel source and destination addresses reversed. The interface address will be set to 192.168.1.2/30, and the interface will also be named gr-0/0/0 but with a sub interface of 0:

```
edit interface gr-0/0/0.0
set tunnel source 192.0.2.10
set tunnel destination 203.0.113.16
set family inet address 192.168.1.2/30
```

If you left the configuration as it is, you will probably start suffering from packet loss due to packets exceeding the MTU and therefore being dropped. Therefore, the first thing you should do is tell Junos OS to turn the "DF" bit off. In addition, you should also tell Junos OS to allow fragmentation on tunneled traffic. Assuming you are at the top of the configuration hierarchy, turning the DF bit off is achieved by running the following command on the HQ SRX:

```
set interface gr-0/0/0.10 clear-dont-fragment-bit
set interface gr-0/0/0.10 tunnel allow-fragmentation
```

On the branch SRX, you would change the command due to the different interface name:

```
set interface gr-0/0/0.0 clear-dont-fragment-bit
set interface gr-0/0/0.0 tunnel allow-fragmentation
```

In addition to clearing the DF bit, one should also attempt to prevent the packet from exceeding the MTU in the first place. This can be done by increasing the MTU itself. Juniper recommends setting the MTU to 1524 bytes, which, if you recall, is the default MTU size of 1500 bytes plus the 24 bytes of the tunnel header. This is set within the interface configuration hierarchy under `family inet`:

```
set interface gr-0/0/0.10 family inet mtu 1524
```

And the branch would be configured as follows:

```
set interface gr-0/0/0.0 family inet mtu 1524
```

Finally, Junos OS also has a rather useful feature called path maximum transmission unit discovery (path MTU discovery), which is a method that allows a Junos OS based device to find out what the lowest MTU is set to in a network path. So when the SRX sends a packet to the destination, it will deliberately turn the DF bit on. This seems counterproductive when you have specified that the DF bit should be turned off, but Junos OS does this for a very good reason as shown in Figure 5.4.

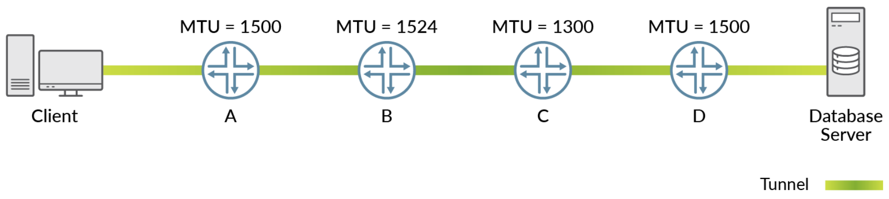


Figure 5.4 Four Routers With Different MTUs

If you look at Figure 5.4, you will notice a client and a server and in-between a network path consisting of four routers. Above each router is number representing the value of the MTU that the router interfaces are set to. Router A has path MTU discovery enabled and it will send a packet and then listen for an *ICMP Fragmentation Needed* message.

Router B has its interfaces set with an MTU of 1524 so it will just forward the packet on. Router C's interfaces, however, have an MTU of 1300. This will be smaller than the packet sent by router A, therefore router C will respond with the *Fragmentation Needed* message router A was waiting for. Router A will automatically adjust the MTU accordingly and in addition, periodically send a probe to see if the MTU has increased or not.

The way you would enable path MTU discovery on the HQ SRX is to run the following command:

```
set interface gr-0/0/0.10 tunnel path-mtu-discovery
```

On the branch SRX, you run the same command, changing the interface to the one used in the branch:

```
set interface gr-0/0/0.10 tunnel path-mtu-discovery
```

NOTE One caveat with path MTU discovery is occasionally administrators prevent network devices from sending ICMP messages. In that situation, the router will simply need to revert back to fragmentation.

In theory, that's all you need to do in order to try to prevent packets from being fragmented or to allow them to be fragmented if absolutely necessary. In either case, this should stop packets from being dropped due to them exceeding the MTU size. You should, however, add the gr-0/0/0.10 interface on the HQ SRX and gr-0/0/0.0 interface on the branch SRX to the trusted security zone of the SRX, and in addition, add them to OSPF so that they can form adjacencies and exchange routing information.

On both of the SRX Series firewalls, the trusted zone is called TRUST and all interfaces will be in area 0.0.0.0 of OSPF. On the HQ SRX you would add this configuration as follows:

```
set security zones security-zone TRUST interfaces gr-0/0/0.10
set security zones security-zone TRUST interfaces gr-0/0/0.10 host-inbound-traffic protocols ospf
set protocols ospf area 0.0.0.0 interface gr-0/0/0.10
```

When running the command on the branch SRX, you would need to simply change the interface name:

```
set security zones security-zone TRUST interfaces gr-0/0/0.0
set security zones security-zone TRUST interfaces gr-0/0/0.0 host-inbound-traffic protocols ospf

set protocols ospf area 0.0.0.0 interface gr-0/0/0.0
```

All that remains now is to commit the changes and test to see if everything works.

Testing

Once you committed the changes, you need to test to make sure traffic is passing over the GRE tunnel. First ping the IP address of the tunnel interface in the branch from HQ. As the default route is going to an ISP, the ISP will drop any traffic coming from a private IP address. This means if you do have a response, it must have travelled over the GRE tunnel. In our case, you want to check if large packets are allowed, therefore let's set the size of the ping to 1496 bytes:

```
root@ACME-HQ-SRX> ping size 1496 192.168.1.2 count 4
PING 192.168.1.2 (192.168.1.2): 1496 data bytes
1504 bytes from 192.168.1.2: icmp_seq=0 ttl=64 time=4.134 ms
1504 bytes from 192.168.1.2: icmp_seq=1 ttl=64 time=4.638 ms
```

```
1504 bytes from 192.168.1.2: icmp_seq=2 ttl=64 time=3.857 ms
1504 bytes from 192.168.1.2: icmp_seq=3 ttl=64 time=4.212 ms

--- 192.168.1.2 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max/stddev = 3.857/4.210/4.638/0.280 ms
```

This worked great, however, the question is: was this packet fragmented or not? To confirm this question, run the ping again but enable the `do-not-fragment` option:

```
root@ACME-HQ-SRX> ping size 1496 192.168.1.2 count 4 do-not-fragment
PING 192.168.1.2 (192.168.1.2): 1496 data bytes

--- 192.168.1.2 ping statistics ---
4 packets transmitted, 0 packets received, 100% packet loss
```

As you can see the packet was dropped, indicating the initial ping sent was fragmented; but at least it wasn't dropped.

The next thing you should check is that OSPF has formed an adjacency over the tunnel. The `show ospf neighbor` command can easily check this:

```
root@ACME-HQ-SRX> show ospf neighbor
Address          Interface              State     ID               Pri  Dead
192.168.1.2      gr-0/0/0.10            Full      172.23.7.2       128    31
```

Finally, you can also monitor the traffic using the monitor traffic interface gr-0/0/0 command to see if traffic destined for the SRX (therefore OSPF hello packets) is being sent across the tunnel:

```
root@ACME-HQ-SRX> monitor traffic interface gr-0/0/0 no-resolve
verbose output suppressed, use <detail> or <extensive> for full protocol decode
Address resolution is OFF.
Listening on gr-0/0/0, capture size 96 bytes

20:52:59.495444  In IP 192.168.1.2 > 224.0.0.5: OSPFv2, Hello, length 60
20:53:08.135786  In IP 192.168.1.2 > 224.0.0.5: OSPFv2, Hello, length 60
20:53:15.813676  In IP 192.168.1.2 > 224.0.0.5: OSPFv2, Hello, length 60
```

And, as you can clearly see from the output, they are being sent. This means our tunnel is up and passing traffic. In addition, if absolutely necessary, the router will fragment the packet in order to prevent them from being dropped.

Recipe 6: Application-based Routing

By Michel Tepper

- Junos OS Used: 18.1R2.5
- Juniper Platforms General Applicability: SRX, vSRX

This recipe shows how to configure advanced policy-based routing (APBR) based on the application being used by the end user.

Problem

When your SRX has multiple connections to the Internet, you may want to differentiate mission-critical traffic from less important traffic. In this case we want to route social media traffic through a secondary ISP.

Solution

Starting from Junos 15.1X49-D110, the SRX Series can route based on an application definition as opposed to Layer 3 and Layer 4 information. The functionality is part of the Juniper AppSecure feature set and is included with any SRX that has the Secure Edge software. Officially it's called advanced policy-based routing (APBR), but most engineers will talk about *application-based routing* when referring to the feature set.

Let's review Figure 6.1 to illustrate our current network.

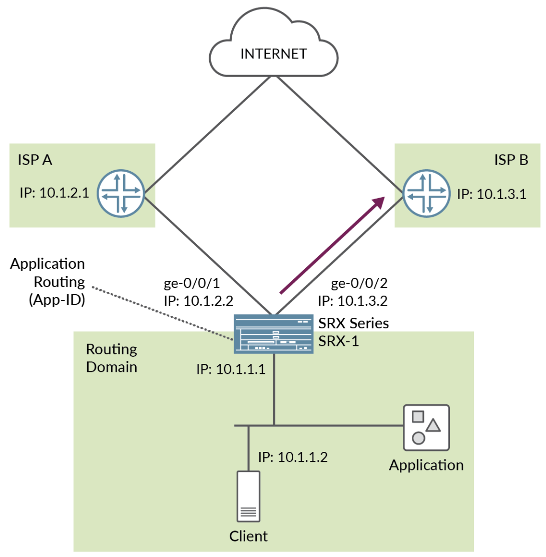


Figure 6.1 Application-based Routing Setup

In Figure 6.1 default routing is done to ISP-A, where all social media traffic should be routed using ISP-B.

SRX-1 shows the following:

```
user@SRX-1> show interfaces terse
......
ge-0/0/1                up    up
ge-0/0/1.0              up    up   inet      10.1.2.2/24
ge-0/0/2                up    up
ge-0/0/2.0              up    up   inet      10.1.3.2/24
......

user@SRX-1> show route
inet.0: 9 destinations, 9 routes (9 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

0.0.0.0/0          *[Static/5] 06:09:55
                    > to 10.1.2.1 via ge-0/0/1.0
.......
```

When you want to route certain traffic to 10.1.3.1 (ISP-B), you need to create a forwarding routing instance:

```
set routing-instances SocMedInstance instance-type forwarding
set routing-instances SocMedInstance routing-options static route 0.0.0.0/0 next-hop 10.1.3.1
set routing-instances SocMedInstance routing-options static route 0.0.0.0/0 qualified-next-
hop 10.1.2.1 metric 10
```

In routing instance `SocMediaInstance` the connected route from ge-0/0/2 is not known yet, so you need to "route leak" the connected route from the instance master to this instance. To do this use the instance import statement in the newly created routing instance and filter out the desired connected route. To create the route filter use this statement:

```
set policy-options policy-statement ImportConnected term 1 from instance master
set policy-options policy-statement ImportConnected term 1 then next term
set policy-options policy-statement ImportConnected term 2 from protocol direct
set policy-options policy-statement ImportConnected term 2 then accept
set policy-options policy-statement ImportConnected term 3 then reject
```

The filter is then applied to the routing instance `SocMediaInstance`:

```
set routing-instances SocMedInstance routing-options instance-import ImportConnected
```

The configuration so far should look like this:

```
user@SRX-1# show routing-instances
SocMedInstance {
    instance-type forwarding;
    routing-options {
        static {
            route 0.0.0.0/0 {
                next-hop 10.1.3.1;
                qualified-next-hop 10.1.2.1 {
                    metric 10;
                }
            }
        }
        instance-import ImportConnected;
    }
}

user@SRX-1# show policy-options
policy-statement ImportConnected {
    term 1 {
        from instance master;
        then next term;
    }
    term 2 {
        from protocol direct;
        then accept;
    }
    term 3 {
        then reject;
    }
```

The relevant parts of the routing table now look like:

```
user@SRX-1> show route


inet.0: 9 destinations, 9 routes (9 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

0.0.0.0/0          *[Static/5] 06:09:55
```

```
                    > to 10.1.2.1 via ge-0/0/1.0
.......
10.1.2.0/24        *[Direct/0] 06:11:03
                    > via ge-0/0/1.0
10.1.2.2/32        *[Local/0] 06:11:03
                      Local via ge-0/0/1.0
10.1.3.0/24        *[Direct/0] 06:11:03
                    > via ge-0/0/2.0
10.1.3.2/32        *[Local/0] 06:11:03
                      Local via ge-0/0/2.0
.......
SocMedInstance.inet.0: 4 destinations, 5 routes (4 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

0.0.0.0/0          *[Static/5] 01:24:59
                    > to 10.1.3.1 via ge-0/0/2.0
                    [Static/5] 01:24:59, metric 10
                    > to 10.1.2.1 via ge-0/0/1.0
10.1.2.0/24        *[Direct/0] 01:24:59
                    > via ge-0/0/1.0
10.1.3.0/24        *[Direct/0] 01:24:59
                    > via ge-0/0/2.0
........
```

Just to recap: we have a forwarding routing instance that sets its route towards ISP-B. The directly connected routes are also present in the forwarding routing instance. Now we can configure APBR to direct traffic to this forwarding routing instance.

With filter-based forwarding (FBF) you would configure a stateless firewall filter to intercept interesting traffic based off of Layer 3 and Layer 4 information, and send it to the forwarding routing instance. However we want to use the App-Route feature to recognize and re-route applications. App-Route is part of the AppSecure suite, and a license is provided if the SRX was purchased with the Secure Edge software.

NOTE SRX1500+ have the AppSecure suite on by default. Same for the SRX300 series if you purchased the Secure Edge software: https://www.juniper.net/documentation/en_US/junos/topics/topic-map/security-application-identification-predefined-signatures.html.

Let's load the AppSecure signatures by first acquiring the license and then running the operational commands to download and install the Application-ID signatures:

```
request system license update
request system services application-identification download
request system services application-identification install
```

Now you need to create an APBR profile to route social networking traffic through ISP-B:

```
set security advance-policy-based-routing profile MyProfile rule social-media match dynamic-
application-group junos:web:social-networking
```

```
set security advance-policy-based-routing profile MyProfile rule social-media then routing-
instance SocMedInstance
```

So now we have an APBR profile that detects social media applications and forwards them to the `SocMedInstance` forwarding routing instance. From there the applications will leave the SRX through ISP-B. The configuration should look like this:

```
user@SRX-1# show security advance-policy-based-routing
profile MyProfile {
    rule social-media {
        match {
            dynamic-application-group junos:web:social-networking;
        }
        then {
            routing-instance SocMedInstance;
        }
    }
}
```

Just having a profile configured won't actually forward traffic to ISP-B. We need to apply the profile based on where traffic comes into to the SRX; in this case we will apply the profile on the trust security zone:

```
Set security zones security-zone trust advance-policy-based-routing-profile MyProfile
```

The configuration should look like:

```
user@SRX-1# show security zones security-zone trust
host-inbound-traffic {
     .........
advance-policy-based-routing-profile {
    MyProfile;
}
```

After the final commit everything should be set. Let's connect a laptop to SRX-1 on the trust side of the SRX and access a social media application like Facebook. All traffic should use ge-0/0/1 as the outgoing interface, but the Facebook traffic should leave SRX-1 on interface ge-0/0/2. Let's verify using the `show security flow session` command:

```
Session ID: 231, Policy name: trust-to-untrust/5, Timeout: 1762, Valid
  In: 10.1.1.2/54050 --> 54.68.233.232/443;tcp, Conn Tag: 0x0, If: irb.0, Pkts: 26, Bytes: 5952,
  Out: 54.68.233.232/443 --> 10.1.2.2/18313;tcp, Conn Tag: 0x0, If: ge-
0/0/1.0, Pkts: 26, Bytes: 15576,
```

So indeed ge-0/0/1 – that is, the path through ISP-A – is still being used as the default path. Let's check the Facebook session with the following: `show security flow session dynamic-application junos:FACEBOOK-ACCESS`.

```
Session ID: 577, Policy name: trust-to-untrust/5, Timeout: 1778, Valid
  In: 10.1.1.2/52769 --> 31.13.92.38/443;tcp, Conn Tag: 0x0, If: irb.0, Pkts: 1065, Bytes: 202471,
  Out: 31.13.92.38/443 --> 10.1.2.2/3559;tcp, Conn Tag: 0x0, If: ge-
```

```
0/0/2.0, Pkts: 865, Bytes: 778279
```

The Facebook session uses ge-0/0/2 as egress interface, so yes: our configuration is working!

You can check this further by using the `statistics` option:

```
user@SRX-1> show security advance-policy-based-routing statistics
  Advance Profile Based Routing statistics:
    Sessions Processed                       794
    AppID cache hits                         652
    AppID requested                          165
    Rule matches                             36
    Route changed on cache hits              0
    Route changed midstream                  36
    Zone mismatch                            0
    Drop on zone mismatch                    0
    Next hop not found                       0
```

And you can see that things are configured correctly.

Discussion

The functionality of advanced policy-based routing (AppRoute) is doing what it's supposed to do. While the routing is now working as we expected, if you take a closer look at the session table you'll see that the return traffic is sent to 10.1.1.2 – the IP address of ge-0/0/1.

This means that the Source NAT IP has not changed when routing is changed mid-session; otherwise the server would receive traffic from a new IP address while handling an existing session. This obviously isn't desirable, but the consequence is that the return traffic will be routed to ge-0/0/1.

By default the SRX will not re-route an existing session towards ISP-B, but any new session will be redirected. If you want to change this behavior the command to do so is below:

```
user@SRX-1> set security advance-policy-based-routing tunables max-route-change 1
```

MORE? Information on this setting can be found in the following KB article: https://kb.juniper.net/InfoCenter/index?page=content&id=KB32303. Additional resources on logging and troubleshooting APBR can be found at: https://www.juniper.net/documentation/en_US/junos/topics/topic-map/security-application-advanced-policy-based-routing.html.

Recipe 7: Automating Junos with Salt

by Peter Klimai

- Salt Version Used: 2018.3 (Oxygen)
- Juniper Platforms General Applicability: All Junos-based platforms
- Junos OS required: 11.4 or later

This recipe will help you to get started automating Junos devices with Salt, the powerful configuration management and remote execution software platform. You will work through Salt installation and basic settings step-by-step, then you will remotely execute some of the related Junos commands and apply some Junos device configurations. This allows you to learn relevant Salt concepts while practicing with it.

Problem

You want to automate certain operational and configuration tasks on your Junos network devices.

Solution

Salt (https://saltstack.com/) is one of the most powerful, scalable, and flexible platforms that allows you automate key operational and configuration tasks, and it already comes with modules supporting Junos OS.

NOTE Yet another reason to start learning and using Salt is its native support for event-driven infrastructure (EDI). With Salt, you can automatically react on certain Junos events in certain ways, as you see fit. Generally, it is recommended to first automate provisioning and monitoring, so this recipe does not go into any EDI details.

NOTE This recipe is intended to get you started. Salt has extensive documentation available at https://docs.saltstack.com.

Setting up Salt and configuring it to work with Junos is quite easy if you follow the steps in this recipe. The terminology used by Salt and the basics of its architecture, as it applies to managing Junos devices, is explained in-line.

The lab setup used for this recipe is shown in Fig. 7.1.

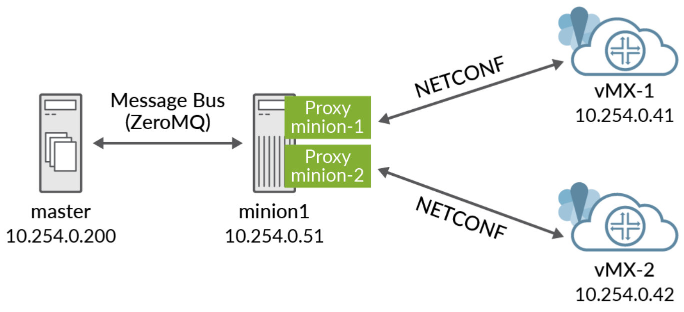


Figure 7.1 *This Recipe's Lab Setup*

The setup includes a Salt Master server (`master`) and a Salt Minion server (`minion1`) running two proxy minions, one for each of two vMX devices in the topology. Management IP addresses are shown.

Some details on the setup are:

- There are two Linux Servers, specifically running Ubuntu 16.04.4 LTS, `master` and `minion1`. Other UNIX flavors should work as well. Servers run as virtual machines using VMWare ESXi hypervisor but you can use any other option, such as KVM or VirtualBox.
- Two Juniper vMX devices, `vMX-1` and `vMX-2`, are also being used.

NOTE *Day One: vMX Up and Running* by Matt Dinham explains the architecture and installation of vMX for KVM hypervisor: https://www.juniper.net/us/en/training/jnbooks/day-one/automation-series/vmx-up-running/. The following blog post by Clay Haynes shows how to run the vMX on VMWare Fusion: https://alostrealist.com/2018/04/16/running-the-vmx-on-vmware-fusion/.

NOTE This recipe will not use any vMX-specific functionality, so replacing the vMXs in this recipe with any other Junos devices, such as physical MX/EX/SRX/QFX Series, or vSRX, will also work for the examples discussed here.

NETCONF over SSH must be enabled on both Junos devices, as follows (this shows for one device – make sure you perform this on both devices):

```
lab@vMX-1> configure
Entering configuration mode

[edit]
lab@vMX-1# set system services netconf ssh

[edit]
lab@vMX-1# commit
commit complete
```

Installing Salt

At this point let's assume you have two Linux servers (VMs) ready for Salt installation, as per Fig. 7.1. When provisioning Linux, it is enough to install standard file system utilities and OpenSSH server.

Here is some key terminology: *Salt Master* is the control server for the main Salt. *Salt Minion* is a machine managed by Salt. Salt is generally agent-based architecture, so devices managed by Salt need to run Salt Minion process. As you will see in the next steps, devices that can't run the Salt Minion process for some reason can still be managed by using proxy minions.

NOTE Generally, Salt is very flexible and every component is customizable and replaceable. The Salt setup may differ greatly depending on the use case – for example, masterless setup is possible. Also, new features are being introduced regularly. In particular, salt-ssh package allows Salt to work in agentless mode (no minion required). This option is not currently used for Junos device management, so it's not discussed here.

The easiest way to set up Salt is through a bootstrap script: (https://docs.saltstack.com/en/latest/topics/tutorials/salt_bootstrap.html). To install Salt on the `master` server, issue the following commands:

```
lab@master:~$ curl -o bootstrap_salt.sh -L https://bootstrap.saltstack.com
lab@master:~$ sudo sh bootstrap_salt.sh -M
```

Output is omitted for brevity. The installation will take a minute or so. When it completes, you can check the Salt version by using the following command:

```
lab@master:~$ salt --version
salt 2018.3.1 (Oxygen)
```

On the `minion1` server, install Salt by performing similar steps, but this time do not use the `-M` key:

```
lab@minion1:~$ curl -o bootstrap_salt.sh -L https://bootstrap.saltstack.com
lab@minion1:~$ sudo sh bootstrap_salt.sh

... OMITTED ...

lab@minion1:~$ salt-minion --version
salt-minion 2018.3.1 (Oxygen)
```

NOTE By default, bootstrap script installs the latest stable Salt version, so it may differ from 2018.3.1 that we use for this recipe. This will typically not be a problem. However, you can enforce installation of the specific Salt version, if you wish, by modifying the second command as follows: `sudo sh bootstrap_salt.sh –M git v2018.3.1` (this is for master; for minion, just omit the `–M` key).

Performing Basic Salt Configuration and Verification

There's one thing you definitely want to configure on the minion, and that's to tell it where the master is:(the default is actually to look for host named "salt", which does not readily meet the needs of this recipe).

So, start configuring the minion on `minion1` server:

```
lab@minion1:~$ sudo vi /etc/salt/minion
```

NOTE Although the examples show you using the vi text editor, any other text editor of your choice will work.

Add the following line (in bold) to the file. (Note that lines starting with a hash are treated as comments):

```
...
# Set the location of the salt master server. If the master server cannot be
# resolved, then the minion will fail to start.
#master: salt
master: 10.254.0.200
...
```

Here, the IP address is the `master` server's address in the lab setup. Replace it with the address in your lab.

Finally, restart the Salt minion process so it re-reads the configuration:

```
lab@minion1:~$ sudo service salt-minion restart
```

Now move to the master and issue the following command to view minion's public key status:

```
lab@master:~$ sudo salt-key --list-all
Accepted Keys:
Denied Keys:
Unaccepted Keys:
minion1.edu.example.com
Rejected Keys:
```

Note the ID of the minion with an unaccepted key. The security system of Salt will not allow communication until you accept minion's key on `master` – so let's do that now:

```
lab@master:~$ sudo salt-key --accept=minion1.edu.example.com
The following keys are going to be accepted:
Unaccepted Keys:
minion1.edu.example.com
Proceed? [n/Y] y
Key for minion minion1.edu.example.com accepted.
```

Now let's execute some Salt commands, using its remote execution capabilities. Note we are testing basic Salt features here - at this point no Junos is involved at all.

First, let's "ping" our minion:

```
lab@master:~$ sudo salt '*' test.ping
minion1.edu.example.com:
    True
```

Here, you called the `test.ping` *execution function* that is used to make sure the minion is up and responding. The communication happens over Salt's ZeroMQ message bus (so this is not an ICMP ping). The argument `'*'` means that you want to execute on all minions, but so far we have only one.

Generally speaking, various Salt *execution modules* allow you to perform (execute) some specific tasks on minions: `test` is just one of such modules.

Let's now try the `cmd.run` function - it allows running arbitrary commands on minions. Let's check minion's Python version:

```
lab@master:~$ sudo salt minion* cmd.run 'python -V'
minion1.edu.example.com:
    Python 2.7.12
```

You can run any other commands on the minions the same way. Check out the full Salt module index at https://docs.saltstack.com/en/latest/salt-modindex.html for more examples.

Enabling Junos Proxy Minion

You do not run Salt minion on-box with Junos devices, instead a proxy minion process is used. The proxy minion process may run either on the master server or on a separate server. The latter option allows for load distribution in scaled setups and seems a bit clearer from the educational point of view, so let's use it here: proxy minions will run on the `minion1` server.

A proxy minion is just a software process (daemon) used to manage, in this case, a networking device. You need one proxy process per device, and for Junos one such process requires about 100MB of RAM, so plan accordingly. Refer again at Figure 7.1. From one "side", proxy minion connects to Salt master using the ZeroMQ bus, while from the other side it is connected to the Junos device using NETCONF protocol (Junos PyEZ library is used under the hood).

To start configuring Salt proxy, edit the `/etc/salt/proxy` file on `minion1` server:

```
lab@minion1:~$ sudo vi /etc/salt/proxy
```

And add a new line to it, telling where the master is:

```
master: 10.254.0.200
```

Before proceeding, it's time to get familiar with one more Salt concept: *pillar*. The pillar system provides various data associated with minions. In the simplest case, pillar files will be YAML files with defined variable values, but pillar data can also be stored in a database such as SQL, obtained via REST API from some external system, etc.

The location where pillar files are stored can vary. By default, it is in `/srv/pillar` directory of the master server (this is defined by `pillar_roots` parameter in `/etc/salt/master` configuration file on the Salt master). Let's just use the default directory – to do so, you will have to create it first:

```
lab@master:~$ sudo mkdir /srv/pillar
```

In this directory, create the `/srv/pillar/proxy-1.sls` file with the following content (just replace host IP, username, and password with values matching your setup, if they are different):

```
lab@master:~$ cat /srv/pillar/proxy-1.sls
proxy:
  proxytype: junos
  host: 10.254.0.41
  username: lab
  password: lab123
  port: 830
```

NOTE Salt certainly has ways to better secure your passwords, but that is beyond the scope of this beginning Salt recipe.

Similarly, create the `/srv/pillar/proxy-2.sls` file:

```
lab@master:~$ cat /srv/pillar/proxy-2.sls
proxy:
  proxytype: junos
  host: 10.254.0.42
  username: lab
  password: lab123
  port: 830
```

The two files that you just created essentially contain some mappings (pairs of keys and corresponding values). For example, the key `username` maps to value `lab`, etc. The key `proxy` has a nested mapping as a value, which is shown by indentation. The format that you just used for these files is YAML (http://yaml.org/), while the file extension is SLS (SaLt State).

Generally, SLS files can be in various formats: in the simplest case it is YAML, or it can be YAML+Jinja (where Jinja is a templates format – see http://jinja.pocoo.org/), or something else if properly customized (maybe even Python code – Salt is very flexible).

Now let's create the *pillar top* file. This file will define which minions have access to which pillar data. In our case, the content will be as follows:

```
lab@master:~$ cat /srv/pillar/top.sls
base:
  'vMX-1':
    - proxy-1
  'vMX-2':
    - proxy-2
```

Here, `base` is the name of what is called *environment* in Salt. For example, you can have testing/staging/production environments – but in this recipe we will only use the default base environment.

Now it's time to perform settings on the minion side. Switch to the `minion1` server. Remember, this server will host a couple of Junos proxy minion processes. For Junos proxy to successfully communicate with Junos devices, a couple of Python packages are needed – namely, `Junos PyEZ` and `jxmlease` (and their dependencies as well). To install those libraries, first install the Python PIP tool, and then the packages themselves (output omitted for brevity):

```
lab@minion1:~$ sudo apt-get install python-pip

lab@minion1:~$ sudo pip install junos-eznc

lab@minion1:~$ sudo python -m easy_install --upgrade pyOpenSSL

lab@minion1:~$ sudo pip install jxmlease
```

NOTE Upgrade the `pyopenssl` package *before* installing `jxmlease` is used here as a workaround, otherwise you may see an error message like this: "`AttributeError: 'module' object has no attribute 'SSL_ST_INIT'`".

Okay it's time to launch the Junos Salt proxy processes:

```
lab@minion1:~$ sudo salt-proxy --proxyid=vMX-1 -d
lab@minion1:~$ sudo salt-proxy --proxyid=vMX-2 -d
```

And, on the master, accept the minion keys, just as you did before:

```
lab@master:~$ sudo salt-key -a vMX-1
The following keys are going to be accepted:
Unaccepted Keys:
vMX-1
Proceed? [n/Y] y
Key for minion vMX-1 accepted.
lab@master:~$ sudo salt-key -a vMX-2
The following keys are going to be accepted:
```

```
Unaccepted Keys:
vMX-2
Proceed? [n/Y] y
Key for minion vMX-2 accepted.
```

How many minions do you think you have now? Let's check with `test.ping`:

```
lab@master:~$ sudo salt '*' test.ping
minion1.edu.example.com:
    True
vMX-1:
    True
vMX-2:
    True
```

So, you have two more minions (proxies) in addition to `minion1`.

Using the Junos Execution Module

Salt includes several execution functions for Junos, such as `junos.cli` (executes the CLI commands and returns the output in the specified format), `junos.install_config` (installs the given configuration), and `junos.install_os` (installs the given software image on the device), and more.

NOTE The complete documentation for the Junos execution module, named `salt.modules.junos`, is available at https://docs.saltstack.com/en/latest/ref/modules/all/salt.modules.junos.html.

Let's first use the `junos.facts` execution function to collect a basic information bundle from our managed devices:

```
lab@master:~$ sudo salt vMX* junos.facts
vMX-2:
    ----------
    facts:
        ----------
        2RE:
            False
        HOME:
            /var/home/lab
        RE0:
            ----------
            last_reboot_reason:
                Router rebooted after a normal shutdown.
...
vMX-1:
    ----------
    facts:
        ----------
        2RE:
            False
        HOME:
            /var/home/lab
        RE0:
            ----------
            last_reboot_reason:
                Router rebooted after a normal shutdown.
...
```

As you can see, a bunch of information is collected and displayed for each device (output is abbreviated). Needless to say, the tasks on different devices are performed in parallel.

A noteworthy thing about facts collected by `junos.facts` is that they are stored in the Salt grains. *Grains* generally contain data about the managed system, and you can use that data in various places while working with Salt—let's say filtering when running execution functions, for example—as shown below:

```
lab@master:~$ sudo salt -G 'os_family:junos' junos.cli "show interfaces fxp0.0 terse"
vMX-1:
    ----------
    message:

        Interface               Admin Link Proto    Local                 Remote
        fxp0.0                  up    up   inet     10.254.0.41/24
    out:
        True
vMX-2:
    ----------
    message:

        Interface               Admin Link Proto    Local                 Remote
        fxp0.0                  up    up   inet     10.254.0.42/24
    out:
        True
```

In this example the CLI command was only executed on devices that had `os_family` grain equal to `junos` (without that filter Salt would also try to execute the `junos.cli` module on `minion1` – that operation would be unsuccessful).

Junos Configuration Management with Salt

Now it's time to apply some configurations. Let's say you want to configure general infrastructure services on your vMX devices – namely, DNS and NTP. One basic idea that you want to take advantage of with automation systems like Salt is configuration templating. That is, network device feature configuration must be separated from variable data like IP addresses, VLAN numbers, etc.

With Salt, the variable data is naturally stored in the pillar system. Let's create a separate file `infrastructure_data.sls` in the pillar root directory, containing lists with all the NTP and DNS servers that you use:

```
lab@master:~$ cat /srv/pillar/infrastructure_data.sls
ntp_servers:
 - 192.168.0.250
 - 192.168.0.251
dns_servers:
 - 192.168.0.253
 - 192.168.0.254
```

This SLS file uses YAML syntax that should, for the most part, already be familiar to you. Dashes represent list elements (a list is just a number of ordered values, such as IP addresses in this example). Remember that indentation is important as it shows structure (nesting of objects) in YAML.

Then, you want to allow your proxy minions to use the data from the `infrastructure_data.sls` file. To do so, edit the pillar top file as follows (new lines that you need to add are in bold; realize there are really many other ways you could achieve the same result):

```
lab@master:~$ cat /srv/pillar/top.sls
base:
  'vMX-1':
    - proxy-1
    - infrastructure_data
  'vMX-2':
    - proxy-2
    - infrastructure_data
```

And also refresh the pillar data as follows:

```
lab@master:~$ sudo salt vMX* saltutil.refresh_pillar
vMX-1:
    True
vMX-2:
    True
```

Now let's create a configuration template – but you need to figure out where to place it first. Salt has a concept of *file roots* directory (actually, it could be a list of directories). It's configured as `file_roots` parameter in the `/etc/salt/master` configuration file on the Salt master, and this location is `/srv/salt` by default, so let's just use it for this recipe. Create the directory as follows:

```
lab@master:~$ sudo mkdir /srv/salt
```

The important thing about file roots is that Salt runs a lightweight file server over the ZeroMQ bus, and minions have access to the files. So placing template files in file roots directories automatically allows minions to read them, which is what you want.

Now back to the template. There are multiple options for how you can create it: Junos text configuration, XML, or set commands. In this example let's create a text configuration template as follows:

```
lab@master:~$ cat /srv/salt/infrastructure_config.conf
system {
  replace: name-server {
{%- for dns_server in pillar.dns_servers %}
  {{ dns_server }};
{%- endfor %}
  }
  replace: ntp {
{%- for ntp_server in pillar.ntp_servers %}
```

```
    server {{ ntp_server }};
{%- endfor %}
  }
}
```

Let's pause to clarify a few points here:

- The file is placed in the file roots directory on the master. It will be downloaded by minions as needed.
- The file extension is `conf`, which will tell the Junos state module that the configuration should be treated as text format.
- Because of the `replace:` tag, you are removing the previously existing DNS and NTP configurations (if any) from the devices. This approach to configuration can be called "declarative" because this way you unambiguously define what exactly will be in those configuration stanzas after change is applied. Alternatively you could do merge load, so other DNS or NTP servers configured previously would remain in configuration.
- You use Jinja syntax for loops (`for – endfor` keywords) and variable value substitution (double curly braces `{{ }}` around a variable).
- Variables—namely lists of servers that are stored in pillar files—are accessed as `pillar.var_name`.

The next step is to create a *state SLS* file. This file will describe what state you want your network devices to be in. In this file, you will use the Junos *state module* (named `salt.states.junos`), in particular its function `install_config` to provision the configuration template.

NOTE Functions of the execution modules could also be used to load Junos device configurations, but Salt states are much more powerful.

Now let's create the state SLS file in the files root directory on the master as follows:

```
lab@master:~$ cat /srv/salt/provision_infrastructure.sls
Install the infrastructure services config:
  junos.install_config:
   - name: salt:///infrastructure_config.conf
   - replace: True
   - timeout: 100
```

In this file:

- `Install the infrastructure services config` is a state name (essentially, an arbitrary string).
- `junos.install_config` is a state function from a `salt.states.junos` state module, the job of this function is to load and commit configurations on Junos devices.

- `name` refers to the path where the configuration (template) file is located.
- `replace: True` specifies that the configuration file uses `replace:` statements.
- `timeout` is NETCONF RPC timeout in seconds. It is especially relevant for commands that take a while to execute.

MORE? The documentation on Salt state modules for Junos is available at: https://docs.saltstack.com/en/latest/ref/states/all/salt.states.junos.html.

Finally, to apply the configuration state described in the SLS file that you just created, you need to execute a `state.apply` function (in this example, vMX devices initially have no NTP or DNS configuration):

```
lab@master:~$ sudo salt vMX* state.apply provision_infrastructure
vMX-2:
----------
          ID: Install the infrastructure services config
    Function: junos.install_config
        Name: salt:///infrastructure_config.conf
      Result: True
     Comment:
     Started: 02:30:16.026562
    Duration: 8464.559 ms
     Changes:
              ----------
              message:
                  Successfully loaded and committed!
              out:
                  True

Summary for vMX-2
------------
Succeeded: 1 (changed=1)
Failed:    0
------------
Total states run:     1
Total run time:   8.465 s
vMX-1:
----------
          ID: Install the infrastructure services config
    Function: junos.install_config
        Name: salt:///infrastructure_config.conf
      Result: True
     Comment:
     Started: 02:30:16.024659
    Duration: 8818.0 ms
     Changes:
              ----------
              message:
                  Successfully loaded and committed!
              out:
                  True
```

```
Summary for vMX-1
------------
Succeeded: 1 (changed=1)
Failed:    0
------------
Total states run:     1
Total run time:   8.818 s
```

For both vMX devices you can see reports that configuration was successfully loaded and committed – let's check on one of them:

```
lab@vMX-1> show configuration | compare rollback 1
[edit system]
+  name-server {
+      192.168.0.253;
+      192.168.0.254;
+  }
+  ntp {
+      server 192.168.0.250;
+      server 192.168.0.251;
+  }
```

So far, looks good!

And if you applied the same state again, the following would happen:

```
lab@master:~$ sudo salt vMX* state.apply provision_infrastructure
vMX-1:
----------
          ID: Install the infrastructure services config
    Function: junos.install_config
        Name: salt:///infrastructure_config.conf
      Result: True
     Comment:
     Started: 02:33:28.329635
    Duration: 533.32 ms
     Changes:
              ----------
              message:
                  Configuration already applied!
              out:
                  True

Summary for vMX-1
------------
Succeeded: 1 (changed=1)
Failed:    0
------------
Total states run:     1
Total run time: 533.320 ms
vMX-2:
----------
          ID: Install the infrastructure services config
    Function: junos.install_config
        Name: salt:///infrastructure_config.conf
      Result: True
     Comment:
     Started: 02:33:28.406416
    Duration: 463.954 ms
     Changes:
              ----------
              message:
                  Configuration already applied!
              out:
                  True

Summary for vMX-2
------------
Succeeded: 1 (changed=1)
Failed:    0
------------
Total states run:     1
Total run time: 463.954 ms
```

As nothing needed to be changed on the devices, no configuration change happened, and no commit was performed, which is natural.

Discussion

This recipe aimed to show you the basics of how Salt works with Junos OS, using specialized Junos modules that give you all the flexibility you need when managing these devices.

Salt also includes NAPALM (http://napalm.readthedocs.io/en/latest/) modules support. NAPALM stands for Network Automation and Programmability Abstraction Layer with Multivendor support and allows you to manage Junos, as well as the equipment of other vendors, using an uniform interface. As normally happens when abstraction layers are added, with NAPALM your work becomes more high level (you don't have to think of fine details, such as vendor-specific configurations), but you also lose some flexibility, compared to working with specialized vendor modules. Choose the ones that best fit your needs.

Hopefully, this recipe helped you to grasp the basics of Salt. For more details, visit the Salt Documentation site at https://docs.saltstack.com. It includes a lot of clear tutorials and great examples. My new book (coming soon, and listed below) will also help you advance your Salt skills.

EDITOR'S NOTE This recipe has been excerpted from a new *Day One* book by Peter Klimai, *Day One: Automating Junos with Salt,* with a publication date of December 2018. Peter takes the introduction provided here and expands on all the concepts, plus plenty more. See https://www.juniper.net/dayone.

Recipe 8: Configuring EVPN Anycast Gateway with Intra-Tenant Inter-Subnet Routing

By Dan Hearty

- Junos OS Used: 15.1X53-D60.4
- Juniper Platforms General Applicability: QFX Series

Problem

Two hosts are members of the same tenant but reside in different subnets and there is a requirement to provide a Layer 3 gateway in order for the two hosts to communicate with one another. They must also retain IP connectivity with one another in the event of a gateway failure. The hosts are connected via different leaf switches in an EVPN-VXLAN fabric. How do you provide redundant L3 Gateway in an EVPN-VXLAN fabric with inter-tenant intra-subnet connectivity?

Solution

EVPN Anycast Gateway with inter-subnet routing techniques can be implemented to help solve this problem. But first a little background.

During the initial conception of EVPN Layer 3 gateways, it was assumed that all PE devices would be configured with a Layer 3 interface (IRB) for a given virtual network. It was also intended that all IRB interfaces would be configured with the same IP address, thus creating a redundant gateway mechanism. This worked great until EVPN-VXLAN came along, and the hardware that was being deployed at the leaf layer no longer provided crucial support for VXLAN L3 Gateway (IRB). As a result, anycast gateway, or virtual gateway address, was created to overcome this limitation.

Typically an EVPN anycast gateway is deployed at the spine layer in an IP fabric architecture. However, recent switch hardware, such as the QFX5110 (Broadcom Trident II+) from Juniper Networks, now supports Layer 3 VXLAN gateways, meaning anycast gateway can be deployed at the leaf. This provides some benefits such as eliminating the need to route traffic up to the spine when routing between tenant subnets.

EVPN anycast gateway works in a fashion very similar to VRRP, whereby a virtual IP and a virtual MAC are used by hosts to forward traffic out of a local virtual network. The significant difference, however, is that EVPN is all-active by design, meaning traffic can be processed by any switch that is configured with a Virtual Gateway Address (VGA). VRRP can only support a single gateway in a given cluster. EVPN type-1 and type-2 routes are significant and provide leaf devices with the information they need to forward traffic towards the gateway.

So, in order to do this, it's necessary to provide each host with a Layer 3 gateway and to ensure that each Layer 3 gateway is redundant. Tenant3 also needs to be enabled for inter-subnet routing. To meet these requirements we're going to implement EVPN anycast gateway on our vQFX10k spine switches. EVPN anycast gateway is enabled via the virtual-gateway-address feature.

As shown in Figure 8.1, host T3-1 is connected to DC1-LEAF1 in subnet 192.168.31.0/24. There is another host, T3-8, that is connected to DC1-LEAF2 in subnet 192.168.38.0/24.

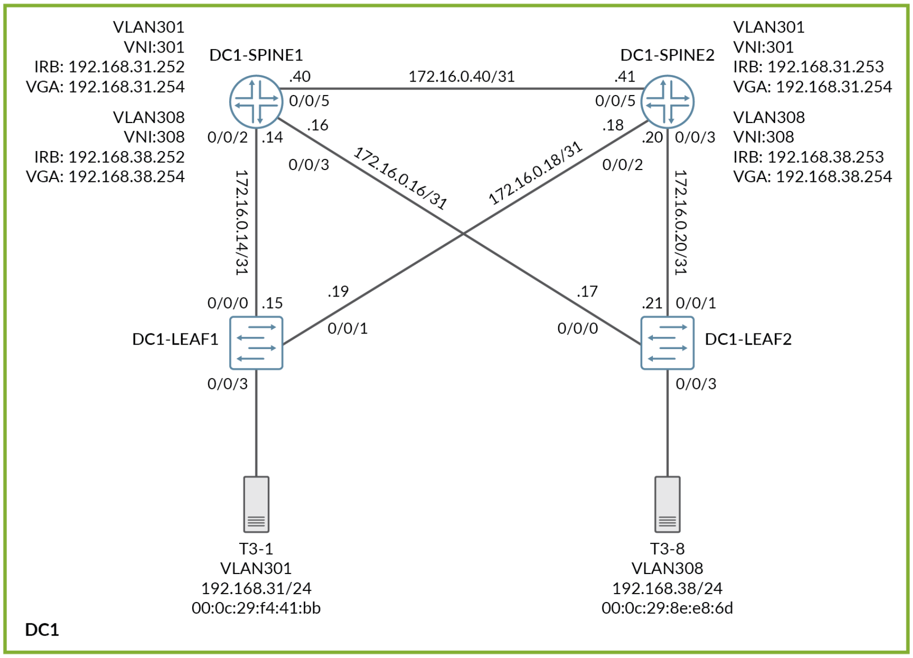


Figure 8.1 EVPN-VXLAN Lab Topology

The network is based on an EVPN-VXLAN 3-stage CLOS IP-Fabric. BGP is used in both the underlay network and the overlay network as described in the following sections.

Underlay Network

The underlay network that is used in this solution is based on an ECMP IP-fabric utilizing EBGP. There are a number of plausible options for the underlay. An IGP such as OSPF or ISIS could be used, for example, or you might also consider using BGP. In fact, whether you are working with a very large deployment or a small four-node solution, such as the one in this recipe, BGP serves as a very good option for the underlay. More specifically, EBGP is a popular choice given the inherent behavior of the protocol. The configuration statements have been omitted for the underlay in this recipe as it bears little relevance to the EVPN anycast gateway solution.

Overlay Network

The overlay network is based on EVPN with VXLAN overlay tunnels. IBGP is used for signaling EVPN in a route-reflector architecture, whereas the spine switches act as BGP route reflectors and the leaf switches act as BGP route reflector clients. IBGP with route reflection makes the solution very scalable, and additional leaf nodes can be added with relative ease (this same argument can effectively be made in a service provider environment with regard to PE scaling).

DC1 Spine Switches

To kick things off let's start with the configuration elements required to enable the EVPN overlay network on the spine switches. This includes IBGP route reflection, IBGP peering between spine nodes, EVPN, and global switch-options.

Configure RR IBGP Overlay for Spine Switches

Configure the spine switches with IBGP towards the leaf switches. As the spine switches are providing route reflector functionality, you must enable a cluster-ID on both spine switches. This BGP peering is specifically dedicated for the overlay, thus you only need to enable family EVPN.

DC1-SPINE1

```
lab@DC1-SPINE1> show configuration protocols bgp group overlay-evpn-rr
type internal;
local-address 10.0.255.1;
family evpn {
    signaling;
}
vpn-apply-export;
cluster 1.1.1.1;
local-as 65001;
multipath;
```

```
neighbor 10.0.255.3;
neighbor 10.0.255.4;

lab@DC1-SPINE1> show configuration protocols bgp group overlay-evpn-rr | display set
set protocols bgp group overlay-evpn-rr type internal
set protocols bgp group overlay-evpn-rr local-address 10.0.255.1
set protocols bgp group overlay-evpn-rr family evpn signaling
set protocols bgp group overlay-evpn-rr vpn-apply-export
set protocols bgp group overlay-evpn-rr cluster 1.1.1.1
set protocols bgp group overlay-evpn-rr local-as 65001
set protocols bgp group overlay-evpn-rr multipath
set protocols bgp group overlay-evpn-rr neighbor 10.0.255.3
set protocols bgp group overlay-evpn-rr neighbor 10.0.255.4
```

DC1-SPINE2

```
lab@DC1-SPINE2> show configuration protocols bgp group overlay-evpn-rr
type internal;
local-address 10.0.255.2;
family evpn {
    signaling;
}
vpn-apply-export;
cluster 1.1.1.1;
local-as 65001;
multipath;
neighbor 10.0.255.3;
neighbor 10.0.255.4;

lab@DC1-SPINE2> show configuration protocols bgp group overlay-evpn-rr | display set
set protocols bgp group overlay-evpn-rr type internal
set protocols bgp group overlay-evpn-rr local-address 10.0.255.2
set protocols bgp group overlay-evpn-rr family evpn signaling
set protocols bgp group overlay-evpn-rr vpn-apply-export
set protocols bgp group overlay-evpn-rr cluster 1.1.1.1
set protocols bgp group overlay-evpn-rr local-as 65001
set protocols bgp group overlay-evpn-rr multipath
set protocols bgp group overlay-evpn-rr neighbor 10.0.255.3
set protocols bgp group overlay-evpn-rr neighbor 10.0.255.4
```

Configure iBGP Overlay for Peer Spine Switch

Here you'll configure another BGP group that looks pretty much identical to the previous group, except this time you don't need to enable a cluster-ID as this is the session between the spine switches.

DC1-SPINE1

```
lab@DC1-SPINE1> show configuration protocols bgp group overlay-evpn
type internal;
local-address 10.0.255.1;
family evpn {
    signaling;
}
local-as 65001;
multipath;
neighbor 10.0.255.2;
```

```
lab@DC1-SPINE1> show configuration protocols bgp group overlay-evpn | display set
set protocols bgp group overlay-evpn type internal
set protocols bgp group overlay-evpn local-address 10.0.255.1
set protocols bgp group overlay-evpn family evpn signaling
set protocols bgp group overlay-evpn local-as 65001
set protocols bgp group overlay-evpn multipath
set protocols bgp group overlay-evpn neighbor 10.0.255.2
```

DC1-SPINE2

```
lab@DC1-SPINE2> show configuration protocols bgp group overlay-evpn
type internal;
local-address 10.0.255.2;
family evpn {
    signaling;
}
local-as 65001;
multipath;
neighbor 10.0.255.1;

lab@DC1-SPINE2> show configuration protocols bgp group overlay-evpn | display set
set protocols bgp group overlay-evpn type internal
set protocols bgp group overlay-evpn local-address 10.0.255.2
set protocols bgp group overlay-evpn family evpn signaling
set protocols bgp group overlay-evpn local-as 65001
set protocols bgp group overlay-evpn multipath
set protocols bgp group overlay-evpn neighbor 10.0.255.1
```

Configure EVPN

Next configure the initial EVPN parameters required in order to build the overlay. Additional parameters will be added to this section when we come to enabling the two tenant subnets.

DC1-SPINE1

```
lab@DC1-SPINE1> show configuration protocols evpn
encapsulation vxlan;
multicast-mode ingress-replication;
default-gateway no-gateway-community;

lab@DC1-SPINE1> show configuration protocols evpn | display set
set protocols evpn encapsulation vxlan
set protocols evpn multicast-mode ingress-replication
set protocols evpn default-gateway no-gateway-community
```

DC1-SPINE2

```
lab@DC1-SPINE2> show configuration protocols evpn
encapsulation vxlan;
multicast-mode ingress-replication;
default-gateway no-gateway-community;

lab@DC1-SPINE2> show configuration protocols evpn | display set
set protocols evpn encapsulation vxlan
set protocols evpn multicast-mode ingress-replication
set protocols evpn default-gateway no-gateway-community
```

Configure Switch-options

Next we need to configure the default switch instance to support VXLAN, define the EVPN import policy, and assign the VRF target that is used for EVPN type-1 routes.

NOTE Type-2 and type-3 EVPN routes are tagged with the community defined under protocols EVPN when enabling the tenant.

DC1-SPINE1

```
lab@DC1-SPINE1> show configuration switch-options
vtep-source-interface lo0.0;
route-distinguisher 10.0.255.1:1;
vrf-import EVPN_IMPORT;
vrf-target target:9999:9999;

lab@DC1-SPINE1> show configuration switch-options | display set
set switch-options vtep-source-interface lo0.0
set switch-options route-distinguisher 10.0.255.1:1
set switch-options vrf-import EVPN_IMPORT
set switch-options vrf-target target:9999:9999
```

DC1-SPINE2

```
lab@DC1-SPINE2> show configuration switch-options
vtep-source-interface lo0.0;
route-distinguisher 10.0.255.2:1;
vrf-import EVPN_IMPORT;
vrf-target target:9999:9999;

lab@DC1-SPINE2> show configuration switch-options | display set
set switch-options vtep-source-interface lo0.0
set switch-options route-distinguisher 10.0.255.2:1
set switch-options vrf-import EVPN_IMPORT
set switch-options vrf-target target:9999:9999
```

Configure EVPN Import Policy and ESI Community

This policy is used to define what is can be imported into the EVPN instance.

NOTE The ESI community defined below is the same as the `vrf-target` (`target:9999:9999`) that was defined in the previous step. This community is used for all type-1 EVPN routes.

DC1-SPINE1

EVPN Import Policy

```
lab@DC1-SPINE1> show configuration policy-options policy-statement EVPN_IMPORT
term import_ESI {
    from community ESI;
    then accept;
}
term last {
    then reject;
}

lab@DC1-SPINE1> show configuration policy-options policy-statement EVPN_IMPORT | display set
set policy-options policy-statement EVPN_IMPORT term import_ESI from community ESI
set policy-options policy-statement EVPN_IMPORT term import_ESI then accept
set policy-options policy-statement EVPN_IMPORT term last then reject
```

ESI Community

```
lab@DC1-SPINE1> show configuration policy-options community ESI
members target:9999:9999;

lab@DC1-SPINE1> show configuration policy-options community ESI | display set
set policy-options community ESI members target:9999:9999
```

DC1-SPINE2

EVPN Import Policy

```
lab@DC1-SPINE2> show configuration policy-options policy-statement EVPN_IMPORT
term import_T1-1 {
term import_ESI {
    from community ESI;
    then accept;
}
term last {
    then reject;
}

lab@DC1-SPINE2> show configuration policy-options policy-statement EVPN_IMPORT | display set
set policy-options policy-statement EVPN_IMPORT term import_ESI from community ESI
set policy-options policy-statement EVPN_IMPORT term import_ESI then accept
set policy-options policy-statement EVPN_IMPORT term last then reject
```

ESI Community

```
lab@DC1-SPINE2> show configuration policy-options community ESI
members target:9999:9999;

lab@DC1-SPINE2> show configuration policy-options community ESI | display set
set policy-options community ESI members target:9999:9999
```

DC1 Leaf Switches

Now let’s move to the leaf switches and enable the equivalent feature set as configured on the spine switches.

Configure iBGP Overlay to Spine Switches

Here we're configuring the leaf switches with IBGP towards the spines. There's nothing special here other than the only address family required is family EVPN.

DC1-LEAF1

```
lab@DC1-LEAF1> show configuration protocols bgp group overlay-evpn
type internal;
local-address 10.0.255.3;
family evpn {
    signaling;
}
local-as 65001;
multipath;
neighbor 10.0.255.1;
neighbor 10.0.255.2;

lab@DC1-LEAF1> show configuration protocols bgp group overlay-evpn | display set
set protocols bgp group overlay-evpn type internal
set protocols bgp group overlay-evpn local-address 10.0.255.3
set protocols bgp group overlay-evpn family evpn signaling
set protocols bgp group overlay-evpn local-as 65001
set protocols bgp group overlay-evpn multipath
set protocols bgp group overlay-evpn neighbor 10.0.255.1
set protocols bgp group overlay-evpn neighbor 10.0.255.2
```

DC1-LEAF2

```
lab@DC1-LEAF2> show configuration protocols bgp group overlay-evpn
type internal;
local-address 10.0.255.4;
family evpn {
    signaling;
}
local-as 65001;
multipath;
neighbor 10.0.255.1;
neighbor 10.0.255.2;

lab@DC1-LEAF2> show configuration protocols bgp group overlay-evpn | display set
set protocols bgp group overlay-evpn type internal
set protocols bgp group overlay-evpn local-address 10.0.255.4
set protocols bgp group overlay-evpn family evpn signaling
set protocols bgp group overlay-evpn local-as 65001
set protocols bgp group overlay-evpn multipath
set protocols bgp group overlay-evpn neighbor 10.0.255.1
set protocols bgp group overlay-evpn neighbor 10.0.255.2
```

Configure EVPN

Now configure the initial EVPN parameters required to build the overlay. Again, we'll be adding additional parameters to this section once we get to enabling the tenant subnets.

DC1-LEAF1

```
lab@DC1-LEAF1> show configuration protocols evpn
encapsulation vxlan;
multicast-mode ingress-replication;

lab@DC1-LEAF1> show configuration protocols evpn | display set
set protocols evpn encapsulation vxlan
set protocols evpn multicast-mode ingress-replication
```

DC1-LEAF2

```
lab@DC1-LEAF2> show configuration protocols evpn
encapsulation vxlan;
multicast-mode ingress-replication;

lab@DC1-LEAF2> show configuration protocols evpn | display set
set protocols evpn encapsulation vxlan
set protocols evpn multicast-mode ingress-replication
```

Configure Switch-options

Next up is to configure the default switch instance of the leaf switch to support VXLAN, define the EVPN import policy, and assign the VRF target that is used for EVPN Type-1 routes.

DC1-LEAF1

```
lab@DC1-LEAF1> show configuration switch-options
vtep-source-interface lo0.0;
route-distinguisher 10.0.255.3:1;
vrf-import EVPN_IMPORT;
vrf-target target:9999:9999;

lab@DC1-LEAF1> show configuration switch-options | display set
set switch-options vtep-source-interface lo0.0
set switch-options route-distinguisher 10.0.255.3:1
set switch-options vrf-import EVPN_IMPORT
set switch-options vrf-target target:9999:9999
```

DC1-LEAF2

```
lab@DC1-LEAF2> show configuration switch-options
vtep-source-interface lo0.0;
route-distinguisher 10.0.255.4:1;
vrf-import EVPN_IMPORT;
vrf-target target:9999:9999;

lab@DC1-LEAF2> show configuration switch-options | display set
set switch-options vtep-source-interface lo0.0
set switch-options route-distinguisher 10.0.255.4:1
set switch-options vrf-import EVPN_IMPORT
set switch-options vrf-target target:9999:9999
```

Configure EVPN Import Policy and ESI Community

The EVPN import policy on the leaf switches is identical to the EVPN import policy on the spine switches, so at this stage we're only allowing the community assigned for type-1 EVPN routes. Additional terms will be added later when enabling the tenant overlay.

DC1-LEAF1

EVPN Import Policy

```
lab@DC1-LEAF1> show configuration policy-options policy-statement EVPN_IMPORT
term import_ESI {
    from community ESI;
    then accept;
}
term last {
    then reject;
}

lab@DC1-LEAF1> show configuration policy-options policy-statement EVPN_IMPORT | display set
set policy-options policy-statement EVPN_IMPORT term import_ESI from community ESI
set policy-options policy-statement EVPN_IMPORT term import_ESI then accept
set policy-options policy-statement EVPN_IMPORT term last then reject
```

ESI Community

```
lab@DC1-LEAF1> show configuration policy-options community ESI
members target:9999:9999;

lab@DC1-LEAF1> show configuration policy-options community ESI | display set
set policy-options community ESI members target:9999:9999
```

DC1-LEAF2

EVPN Import Policy

```
lab@DC1-LEAF2> show configuration policy-options policy-statement EVPN_IMPORT
term import_ESI {
    from community ESI;
    then accept;
}
term last {
    then reject;
}

lab@DC1-LEAF2> show configuration policy-options policy-statement EVPN_IMPORT | display set
set policy-options policy-statement EVPN_IMPORT term import_ESI from community ESI
set policy-options policy-statement EVPN_IMPORT term import_ESI then accept
set policy-options policy-statement EVPN_IMPORT term last then reject
```

ESI Community

```
lab@DC1-LEAF2> show configuration policy-options community ESI
members target:9999:9999;
```

```
lab@DC1-LEAF2> show configuration policy-options community ESI | display set
set policy-options community ESI members target:9999:9999
```

Configure Tenant

Now it's time to configure tenant3 as part of the overlay. In this recipe we are using Tenant3 host 1 and host 8, or T3-1 and T3-8. Both T3-1 and T3-8 reside in their own virtual network but are members of the same tenancy.

DC1 Leaf Switches

Let's start off by configuring the leaf switches.

Configure Host Access Interface

On each leaf switch you need to configure the access ports that attach to each of the hosts.

DC1-LEAF1

```
lab@DC1-LEAF1> show configuration interfaces xe-0/0/3
description "t3-1 ens192";
unit 0 {
    family ethernet-switching {
        interface-mode access;
        vlan {
            members T3-1;
        }
    }
}

lab@DC1-LEAF1> show configuration interfaces xe-0/0/3 | display set
set interfaces xe-0/0/3 description "t3-1 ens192"
set interfaces xe-0/0/3 unit 0 family ethernet-switching interface-mode access
set interfaces xe-0/0/3 unit 0 family ethernet-switching vlan members T3-1
```

DC1-LEAF2

```
lab@DC1-LEAF2> show configuration interfaces xe-0/0/3
description "t3-8 ens192";
unit 0 {
    family ethernet-switching {
        interface-mode access;
        vlan {
            members T3-8;
        }
    }
}

lab@DC1-LEAF2> show configuration interfaces xe-0/0/3 | display set
set interfaces xe-0/0/3 description "t3-8 ens192"
set interfaces xe-0/0/3 unit 0 family ethernet-switching interface-mode access
set interfaces xe-0/0/3 unit 0 family ethernet-switching vlan members T3-8
```

Configure Host VLAN and VXLAN Settings

Next let's create the VLAN and assign a virtual network identifier (VNI) to identify the virtual network in the overlay.

NOTE It is only necessary to add the VLAN and VNI for the locally attached network for each given leaf switch.

DC1-LEAF1

```
lab@DC1-LEAF1> show configuration vlans
T3-1 {
    vlan-id 301;
    vxlan {
        vni 301;
        ingress-node-replication;
    }
}

lab@DC1-LEAF1> show configuration vlans | display set
set vlans T3-1 vlan-id 301
set vlans T3-1 vxlan vni 301
set vlans T3-1 vxlan ingress-node-replication
```

DC1-LEAF2

```
lab@DC1-LEAF2> show configuration vlans
T3-8 {
    vlan-id 308;
    vxlan {
        vni 308;
        ingress-node-replication;
    }
}

lab@DC1-LEAF2> show configuration vlans | display set
set vlans T3-8 vlan-id 308
set vlans T3-8 vxlan vni 308
set vlans T3-8 vxlan ingress-node-replication
```

Configure Host-specific EVPN Parameters

In order to add the virtual networks to the overlay you now need to configure some additional parameters to the initial EVPN configuration that was set earlier. Take note that each host is only attached to a single leaf switch, thus you are only required to enable the locally attached virtual network for a given leaf switch.

NOTE The `vrf-target export` option is deprecated in Junos OS release 17.3.

DC1-LEAF1

```
lab@DC1-LEAF1> show configuration protocols evpn
encapsulation vxlan;
extended-vni-list [ 301 ];
```

```
multicast-mode ingress-replication;
vni-options {
    vni 301 {
        vrf-target export target:1:301;
    }
}

lab@DC1-LEAF1> show configuration protocols evpn | display set
set protocols evpn encapsulation vxlan
set protocols evpn extended-vni-list 301
set protocols evpn multicast-mode ingress-replication
set protocols evpn vni-options vni 301 vrf-target export target:1:301
```

DC1-LEAF2

```
lab@DC1-LEAF2> show configuration protocols evpn
encapsulation vxlan;
extended-vni-list [ 308 ];
multicast-mode ingress-replication;
vni-options {
    vni 308 {
        vrf-target export target:1:308;
    }
}

lab@DC1-LEAF2> show configuration protocols evpn | display set
set protocols evpn encapsulation vxlan
set protocols evpn extended-vni-list 308
set protocols evpn multicast-mode ingress-replication
set protocols evpn vni-options vni 308 vrf-target export target:1:308
```

Configure EVPN Import Policy

There are two elements to consider when adding to the EVPN import policy. The first is that you need to add an additional term to the existing policy created earlier. This new term is only relevant to the locally attached virtual network, therefore there is no need to create a term for the virtual network that is attached to the other leaf switch. This new term also needs to be inserted before the last default reject term. The second thing to consider is that you need to create a community and attach it to the locally connected virtual network for each given leaf switch.

DC1-LEAF1

EVPN Import Policy

```
lab@DC1-LEAF1> show configuration policy-options policy-statement EVPN_IMPORT
term import_T3-1 {
    from community T3-1;
    then accept;
}
term import_ESI {
    from community ESI;
    then accept;
}
term last {
    then reject;
}

lab@DC1-LEAF1> show configuration policy-options policy-statement EVPN_IMPORT | display set
set policy-options policy-statement EVPN_IMPORT term import_T3-1 from community T3-1
set policy-options policy-statement EVPN_IMPORT term import_T3-1 then accept
set policy-options policy-statement EVPN_IMPORT term import_ESI from community ESI
set policy-options policy-statement EVPN_IMPORT term import_ESI then accept
set policy-options policy-statement EVPN_IMPORT term last then reject
```

Tenant Community

```
lab@DC1-LEAF1> show configuration policy-options community T3-1
members target:1:301;

lab@DC1-LEAF1> show configuration policy-options community T3-1 | display set
set policy-options community T3-1 members target:1:301
```

DC1-LEAF2

EVPN Import Policy

```
lab@DC1-LEAF2> show configuration policy-options policy-statement EVPN_IMPORT
term import_T3-8 {
    from community T3-8;
    then accept;
}
term import_ESI {
    from community ESI;
    then accept;
}
term last {
    then reject;
}

lab@DC1-LEAF2> show configuration policy-options policy-statement EVPN_IMPORT | display set
set policy-options policy-statement EVPN_IMPORT term import_T3-8 from community T3-8
set policy-options policy-statement EVPN_IMPORT term import_T3-8 then accept
set policy-options policy-statement EVPN_IMPORT term import_ESI from community ESI
set policy-options policy-statement EVPN_IMPORT term import_ESI then accept
set policy-options policy-statement EVPN_IMPORT term last then reject
```

Tenant Community

```
lab@DC1-LEAF2> show configuration policy-options community T3-8
members target:1:308;

lab@DC1-LEAF2> show configuration policy-options community T3-8 | display set
set policy-options community T3-8 members target:1:308
```

DC1 Spine Switches

Configure Host VLAN and VXLAN Settings

Now we need to add the VLAN and VNI for each of the virtual networks. However, this time you add both VLANs and VNIs to both spine switches. The configuration on DC1-SPINE1 and DC1-SPINE2 is identical for this step.

DC1-SPINE1

```
lab@DC1-SPINE1> show configuration vlans
T3-1 {
    vlan-id 301;
    l3-interface irb.301;
    vxlan {
        vni 301;
        ingress-node-replication;
    }
}
T3-8 {
    vlan-id 308;
    l3-interface irb.308;
    vxlan {
        vni 308;
        ingress-node-replication;
    }
}

lab@DC1-SPINE1> show configuration vlans | display set
set vlans T3-1 vlan-id 301
set vlans T3-1 l3-interface irb.301
set vlans T3-1 vxlan vni 301
set vlans T3-1 vxlan ingress-node-replication
set vlans T3-8 vlan-id 308
set vlans T3-8 l3-interface irb.308
set vlans T3-8 vxlan vni 308
set vlans T3-8 vxlan ingress-node-replication
```

DC1-SPINE2

```
lab@DC1-SPINE2> show configuration vlans
T3-1 {
    vlan-id 301;
    l3-interface irb.301;
    vxlan {
        vni 301;
        ingress-node-replication;
    }
}
T3-8 {
    vlan-id 308;
    l3-interface irb.308;
    vxlan {
        vni 308;
        ingress-node-replication;
    }
}

lab@DC1-SPINE2> show configuration vlans | display set
set vlans T3-1 vlan-id 301
set vlans T3-1 l3-interface irb.301
set vlans T3-1 vxlan vni 301
set vlans T3-1 vxlan ingress-node-replication
set vlans T3-8 vlan-id 308
set vlans T3-8 l3-interface irb.308
set vlans T3-8 vxlan vni 308
set vlans T3-8 vxlan ingress-node-replication
```

Configure Host-specific EVPN Parameters

As you did with the leaf switches, you need to add some additional parameters to the initial EVPN configuration that was set earlier. However, this time you need to add both virtual networks on each spine switch as they are both attached to each virtual network.

DC1-SPINE1

```
lab@DC1-SPINE1> show configuration protocols evpn
encapsulation vxlan;
extended-vni-list [ 301 308 ];
multicast-mode ingress-replication;
default-gateway no-gateway-community;
vni-options {
    vni 301 {
        vrf-target export target:1:301;
    }
    vni 308 {
        vrf-target export target:1:308;
    }
}

lab@DC1-SPINE1> show configuration protocols evpn | display set
set protocols evpn encapsulation vxlan
set protocols evpn extended-vni-list 301
set protocols evpn extended-vni-list 308
set protocols evpn multicast-mode ingress-replication
set protocols evpn default-gateway no-gateway-community
set protocols evpn vni-options vni 301 vrf-target export target:1:301
set protocols evpn vni-options vni 308 vrf-target export target:1:308
```

DC1-SPINE2

```
lab@DC1-SPINE2> show configuration protocols evpn
encapsulation vxlan;
extended-vni-list [ 301 308 ];
multicast-mode ingress-replication;
default-gateway no-gateway-community;
vni-options {
    vni 301 {
        vrf-target export target:1:301;
    }
    vni 308 {
        vrf-target export target:1:308;
    }
}

lab@DC1-SPINE2> show configuration protocols evpn | display set
set protocols evpn encapsulation vxlan
set protocols evpn extended-vni-list 301
set protocols evpn extended-vni-list 308
set protocols evpn multicast-mode ingress-replication
set protocols evpn default-gateway no-gateway-community
set protocols evpn vni-options vni 301 vrf-target export target:1:301
set protocols evpn vni-options vni 308 vrf-target export target:1:308
```

Configure EVPN Import Policy

Now let's amend the EVPN import policy. Remember to insert each term before the last default reject term. You also need to create a community for each virtual network on both spine switches.

DC1-SPINE1

EVPN Import Policy

```
lab@DC1-SPINE1> show configuration policy-options policy-statement EVPN_IMPORT
term import_T3-1 {
    from community T3-1;
    then accept;
}
term import_T3-8 {
    from community T3-8;
    then accept;
}
term import_ESI {
    from community ESI;
    then accept;
}
term last {
    then reject;
}

lab@DC1-SPINE1> show configuration policy-options policy-statement EVPN_IMPORT | display set
set policy-options policy-statement EVPN_IMPORT term import_T3-1 from community T3-1
set policy-options policy-statement EVPN_IMPORT term import_T3-1 then accept
set policy-options policy-statement EVPN_IMPORT term import_T3-8 from community T3-8
set policy-options policy-statement EVPN_IMPORT term import_T3-8 then accept
set policy-options policy-statement EVPN_IMPORT term import_ESI from community ESI
set policy-options policy-statement EVPN_IMPORT term import_ESI then accept
set policy-options policy-statement EVPN_IMPORT term last then reject
```

T3-1 Community

```
lab@DC1-SPINE1> show configuration policy-options community T3-1
members target:1:301;

lab@DC1-SPINE1> show configuration policy-options community T3-1 | display set
set policy-options community T3-1 members target:1:301
```

T3-8 Community

```
lab@DC1-SPINE1> show configuration policy-options community T3-8
members target:1:308;

lab@DC1-SPINE1> show configuration policy-options community T3-8 | display set
set policy-options community T3-8 members target:1:308
```

DC1-SPINE1

EVPN Import Policy

```
lab@DC1-SPINE2> show configuration policy-options policy-statement EVPN_IMPORT
term import_T3-1 {
    from community T3-1;
    then accept;
}
term import_T3-8 {
    from community T3-8;
    then accept;
}
term import_ESI {
    from community ESI;
    then accept;
}
term last {
    then reject;
}

lab@DC1-SPINE2> show configuration policy-options policy-statement EVPN_IMPORT | display set
set policy-options policy-statement EVPN_IMPORT term import_T3-1 from community T3-1
set policy-options policy-statement EVPN_IMPORT term import_T3-1 then accept
set policy-options policy-statement EVPN_IMPORT term import_T3-8 from community T3-8
set policy-options policy-statement EVPN_IMPORT term import_T3-8 then accept
set policy-options policy-statement EVPN_IMPORT term import_ESI from community ESI
set policy-options policy-statement EVPN_IMPORT term import_ESI then accept
set policy-options policy-statement EVPN_IMPORT term last then reject
```

T3-1 Community

```
lab@DC1-SPINE2> show configuration policy-options community T3-1
members target:1:301;

lab@DC1-SPINE2> show configuration policy-options community T3-1 | display set
set policy-options community T3-1 members target:1:301
```

T3-8 Community

```
lab@DC1-SPINE2> show configuration policy-options community T3-8
members target:1:308;

lab@DC1-SPINE2> show configuration policy-options community T3-8 | display set
set policy-options community T3-8 members target:1:308
```

Configure EVPN Virtual Gateway Address - Anycast Gateway

Now that both virtual networks are configured, it's time to focus on the main requirement: provide intra-tenant, inter-subnet routing via a redundant anycast gateway.

NOTE No changes or special configurations are required on the leaf switches for this solution.

DC1 Hosts

Configure Host Gateways

To kick things off let's start by configuring each of our hosts. On each host you need to configure a default gateway that points towards the VGA. The address to use is the address that is assigned as the VGA for each given tenant subnet: T3-1 = 192.168.31.254; T3-8 = 192.168.38.254.

Host T3-1

```
lab@t3-1:~$ cat /etc/network/interfaces
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

source /etc/network/interfaces.d/*

# The loopback network interface
auto lo
iface lo inet loopback

auto ens192
iface ens192 inet static
        address 192.168.31.1
        netmask 255.255.255.0
        network 192.168.31.0
        broadcast 192.168.31.255
        gateway 192.168.31.254

lab@t3-1:~$ ifconfig
ens192    Link encap:Ethernet  HWaddr 00:0c:29:f4:41:bb
          inet addr:192.168.31.1  Bcast:192.168.31.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fef4:41bb/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:74150 errors:0 dropped:20 overruns:0 frame:0
          TX packets:76410 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:6796536 (6.7 MB)  TX bytes:6837020 (6.8 M)
```

Host T3-8

```
lab@t3-8:~$ cat /etc/network/interfaces
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

source /etc/network/interfaces.d/*

# The loopback network interface
auto lo
iface lo inet loopback

auto ens192
iface ens192 inet static
        address 192.168.38.1
        netmask 255.255.255.0
        network 192.168.38.0
        broadcast 192.168.38.255
        gateway 192.168.38.254
```

```
lab@t3-8:~$ ifconfig
ens192    Link encap:Ethernet  HWaddr 00:0c:29:8e:e8:6d
          inet addr:192.168.38.1  Bcast:192.168.38.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe8e:e86d/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:15689 errors:0 dropped:0 overruns:0 frame:0
          TX packets:16326 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1416436 (1.4 MB)  TX bytes:1397112 (1.3 MB)
```

DC1 Spine Switches

Configure IRB with VGA for Each Subnet

It's time to configure the VGA feature on the spine switches in order to enable EVPN Anycast Gateway. VGA is configured at the IRB level for each given virtual network. You need to configure an IRB interface for each of the virtual networks on both spine switches. Before configuring `virtual-gateway-address`, you need to assign a unique IP address that resides in the same network. The VGA is shared between the spine switches per virtual network.

NOTE As of Junos OS version 16.1, Juniper introduced the `virtual-gateway-accept-data` knob that allows you to send traffic, such as ICMP, directly to the VGA. This feature is synonymous to the VRRP `accept-data` feature.

DC1-SPINE1

```
lab@DC1-SPINE1> show configuration interfaces irb
unit 301 {
    proxy-macip-advertisement;
    description " * T3 - vlan 301 - vni 301 ";
    family inet {
        address 192.168.31.252/24 {
            virtual-gateway-address 192.168.31.254;
        }
    }
}
unit 308 {
    proxy-macip-advertisement;
    description " * T3 - vlan 308 - vni 308 ";
    family inet {
        address 192.168.38.252/24 {
            virtual-gateway-address 192.168.38.254;
        }
    }
}

lab@DC1-SPINE1> show configuration interfaces irb | display set
set interfaces irb unit 301 proxy-macip-advertisement
set interfaces irb unit 301 description " * T3 - vlan 301 - vni 301 "
set interfaces irb unit 301 family inet address 192.168.31.252/24 virtual-gateway-
address 192.168.31.254
set interfaces irb unit 308 proxy-macip-advertisement
set interfaces irb unit 308 description " * T3 - vlan 308 - vni 308 "
set interfaces irb unit 308 family inet address 192.168.38.252/24 virtual-gateway-
address 192.168.38.254
```

DC1-SPINE2

```
lab@DC1-SPINE2> show configuration interfaces irb
unit 301 {
    proxy-macip-advertisement;
    description " * T3 - vlan 301 - vni 301 ";
    family inet {
        address 192.168.31.253/24 {
            virtual-gateway-address 192.168.31.254;
        }
    }
}
unit 308 {
    proxy-macip-advertisement;
    description " * T3 - vlan 308 - vni 308 ";
    family inet {
        address 192.168.38.253/24 {
            virtual-gateway-address 192.168.38.254;
        }
    }
}

lab@DC1-SPINE2> show configuration interfaces irb | display set
set interfaces irb unit 301 proxy-macip-advertisement
set interfaces irb unit 301 description " * T3 - vlan 301 - vni 301 "
set interfaces irb unit 301 family inet address 192.168.31.253/24 virtual-gateway-
address 192.168.31.254
set interfaces irb unit 308 proxy-macip-advertisement
set interfaces irb unit 308 description " * T3 - vlan 308 - vni 308 "
set interfaces irb unit 308 family inet address 192.168.38.253/24 virtual-gateway-
address 192.168.38.254
```

Configure Tenant VRF

So, now that the IRB interfaces are configured, you need to configure a VRF for the tenant. In theory you could just use the global routing instance, but there's an assumption that multiple tenants will be present, which means that there will be multiple VRFs. The point here is that we're placing both IRB interfaces into the VRF. This is what enables inter-subnet routing within the tenant as both networks become locally attached and routable.

DC1-SPINE1

```
lab@DC1-SPINE1> show configuration routing-instances TENANT_3_VRF
instance-type vrf;
interface irb.301;
interface irb.308;
interface lo0.30;
route-distinguisher 10.0.255.1:30;
vrf-target target:1:300;

lab@DC1-SPINE1> show configuration routing-instances TENANT_3_VRF | display set
set routing-instances TENANT_3_VRF instance-type vrf
set routing-instances TENANT_3_VRF interface irb.301
set routing-instances TENANT_3_VRF interface irb.308
set routing-instances TENANT_3_VRF interface lo0.30
set routing-instances TENANT_3_VRF route-distinguisher 10.0.255.1:30
set routing-instances TENANT_3_VRF vrf-target target:1:300
```

DC1-SPINE2

```
lab@DC1-SPINE2> show configuration routing-instances TENANT_3_VRF
instance-type vrf;
interface irb.301;
interface irb.308;
interface lo0.30;
route-distinguisher 10.0.255.2:30;
vrf-target target:1:300;

lab@DC1-SPINE2> show configuration routing-instances TENANT_3_VRF | display set
set routing-instances TENANT_3_VRF instance-type vrf
set routing-instances TENANT_3_VRF interface irb.301
set routing-instances TENANT_3_VRF interface irb.308
set routing-instances TENANT_3_VRF interface lo0.30
set routing-instances TENANT_3_VRF route-distinguisher 10.0.255.2:30
set routing-instances TENANT_3_VRF vrf-target target:1:300
```

Verification

Now that all of the configuration is in place, it's time to verify. We won't go into great detail here, but you can find very comprehensive explanations, detail, and workable configurations in Deepti Chandra's book *This Week: Data Center Deployment with EVPN/VXLAN* that can be found here: https://www.juniper.net/us/en/training/jnbooks/day-one/data-center-technologies/data-center-deployment-evpn-vxlan/.

Verify IRB Interfaces on Spine Switches

First verify that the IRB interfaces that we configured for VNI 301 and VNI 308 are up on both spine switches. Interestingly you won't see the VGA in this output but you will see the inet address that we assigned to each logical unit. You will also see that each IRB logical unit interface is a member of their respective bridge domains.

DC1-SPINE1

```
lab@DC1-SPINE1> show interfaces irb.301
  Logical interface irb.301 (Index 552) (SNMP ifIndex 558)
    Description:  * T3 - vlan 301 - vni 301
    Flags: Up SNMP-Traps 0x4004000 Encapsulation: ENET2
    Bandwidth: 1000mbps
    Routing Instance: default-switch Bridging Domain: T3-1
    Input packets : 0
    Output packets: 123
    Protocol inet, MTU: 1514
      Flags: Sendbcast-pkt-to-re, Is-Primary
      Addresses, Flags: Is-Preferred Is-Primary
        Destination: 192.168.31/24, Local: 192.168.31.252, Broadcast: 192.168.31.255

lab@DC1-SPINE1> show interfaces irb.308
  Logical interface irb.308 (Index 553) (SNMP ifIndex 547)
    Description:  * T3 - vlan 308 - vni 308
    Flags: Up SNMP-Traps 0x4000 Encapsulation: ENET2
    Bandwidth: 1000mbps
    Routing Instance: default-switch Bridging Domain: T3-8
    Input packets : 0
    Output packets: 119
    Protocol inet, MTU: 1514
      Flags: Sendbcast-pkt-to-re
      Addresses, Flags: Is-Preferred Is-Primary
        Destination: 192.168.38/24, Local: 192.168.38.252, Broadcast: 192.168.38.255
```

DC1-SPINE2

```
lab@DC1-SPINE2> show interfaces irb.301
  Logical interface irb.301 (Index 552) (SNMP ifIndex 558)
    Description:  * T3 - vlan 301 - vni 301
    Flags: Up SNMP-Traps 0x4004000 Encapsulation: ENET2
    Bandwidth: 1000mbps
    Routing Instance: default-switch Bridging Domain: T3-1
    Input packets : 0
    Output packets: 1437
    Protocol inet, MTU: 1514
      Flags: Sendbcast-pkt-to-re, Is-Primary
      Addresses, Flags: Is-Preferred Is-Primary
        Destination: 192.168.31/24, Local: 192.168.31.253, Broadcast: 192.168.31.255

lab@DC1-SPINE2> show interfaces irb.308
  Logical interface irb.308 (Index 553) (SNMP ifIndex 542)
    Description:  * T3 - vlan 308 - vni 308
    Flags: Up SNMP-Traps 0x4000 Encapsulation: ENET2
    Bandwidth: 1000mbps
    Routing Instance: default-switch Bridging Domain: T3-8
    Input packets : 0
    Output packets: 1460
    Protocol inet, MTU: 1514
      Flags: Sendbcast-pkt-to-re
      Addresses, Flags: Is-Preferred Is-Primary
        Destination: 192.168.38/24, Local: 192.168.38.253, Broadcast: 192.168.38.255
```

Verify Type-1 Routes That are Generated on Spine Switches and Advertised to Leaf Switches.

Next let’s verify the VGA that we configured for each virtual network and take a look at how this is passed to the leaf switches.

When a Virtual Gateway is configured, a type-1 EVPN route is generated for the shared segment. Junos generates a random ESI that is used to identify the shared gateway. This type-1 EVPN route is then advertised via BGP to the leaf switches.

A quick way to identify the ESI value is to check the EVPN database on the spine.

In the example below, you can see that VNI 301 has an ESI value of 05:00:00:fe:4f:00:00:01:2d:00 and VNI 308 has an ESI value of 05:00:00:fe:4f:00:00:01:34:00. You can also see the VGA that we assigned earlier.

You should notice that the MAC address used for each of the Virtual Gateways is the same MAC address that is used in VRRP.

DC1-SPINE1

```
lab@DC1-SPINE1> show evpn database
Instance: default-switch
VLAN  VNI  MAC address       Active source                  Timestamp        IP address
      301   00:00:5e:00:01:01  05:00:00:fe:4f:00:00:01:2d:00  May 22 14:40:52  192.168.31.254
      301   00:0c:29:f4:41:bb  10.0.255.3                     May 22 14:41:25  192.168.31.1
      301   02:05:86:71:03:00  10.0.255.2                     May 22 14:41:20  192.168.31.253
      301   02:05:86:71:d1:00  irb.301                        May 22 14:40:53  192.168.31.252
      308   00:00:5e:00:01:01  05:00:00:fe:4f:00:00:01:34:00  May 22 14:40:53  192.168.38.254
      308   00:0c:29:8e:e8:6d  10.0.255.4                     May 22 14:41:24  192.168.38.1
      308   02:05:86:71:03:00  10.0.255.2                     May 22 14:41:20  192.168.38.253
      308   02:05:86:71:d1:00  irb.308                        May 22 14:40:53  192.168.38.252
```

DC1-SPINE2

```
lab@DC1-SPINE2> show evpn database
Instance: default-switch
VLAN  VNI  MAC address       Active source                  Timestamp        IP address
      301   00:00:5e:00:01:01  05:00:00:fe:50:00:00:01:2d:00  May 22 14:41:20  192.168.31.254
      301   00:0c:29:f4:41:bb  10.0.255.3                     May 15 07:31:04  192.168.31.1
      301   02:05:86:71:03:00  irb.301                        May 22 14:41:21  192.168.31.253
      301   02:05:86:71:d1:00  10.0.255.1                     May 22 14:40:53  192.168.31.252
      308   00:00:5e:00:01:01  05:00:00:fe:50:00:00:01:34:00  May 22 14:41:20  192.168.38.254
      308   00:0c:29:8e:e8:6d  10.0.255.4                     May 15 07:38:01  192.168.38.1
      308   02:05:86:71:03:00  irb.308                        May 22 14:41:21  192.168.38.253
      308   02:05:86:71:d1:00  10.0.255.1                     May 22 14:40:53  192.168.38.252
```

Now that you've identified the ESI values for each of the virtual networks, let's take a look at how this is advertised to the leaf switches. You can do this by checking the BGP RIB OUT table with the EVPN ESI tag identified in the previous step.

DC1-SPINE1

```
lab@DC1-SPINE1> show route advertising-protocol bgp 10.0.255.3 evpn-esi-
value 05:00:00:fe:4f:00:00:01:2d:00

bgp.evpn.0: 114 destinations, 198 routes (114 active, 0 holddown, 84 hidden)
  Prefix                  Nexthop              MED     Lclpref    AS path
  1:10.0.255.1:0::050000fe4f0000012d00::FFFF:FFFF/304
*                         Self                         100        I

default-switch.evpn.0: 40 destinations, 52 routes (40 active, 0 holddown, 12 hidden)

__default_evpn__.evpn.0: 6 destinations, 6 routes (6 active, 0 holddown, 0 hidden)
  Prefix                  Nexthop              MED     Lclpref    AS path
  1:10.0.255.1:0::050000fe4f0000012d00::FFFF:FFFF/304
*                         Self                         100        I

{master:0}
lab@DC1-SPINE1> show route advertising-protocol bgp 10.0.255.3 evpn-esi-value 05:00:00:fe:
```

```
4f:00:00:01:34:00

bgp.evpn.0: 114 destinations, 198 routes (114 active, 0 holddown, 84 hidden)
  Prefix                  Nexthop              MED     Lclpref    AS path
  1:10.0.255.1:0::050000fe4f0000013400::FFFF:FFFF/304
*                         Self                         100        I

default-switch.evpn.0: 40 destinations, 52 routes (40 active, 0 holddown, 12 hidden)

__default_evpn__.evpn.0: 6 destinations, 6 routes (6 active, 0 holddown, 0 hidden)
  Prefix                  Nexthop              MED     Lclpref    AS path
  1:10.0.255.1:0::050000fe4f0000013400::FFFF:FFFF/304
*                         Self                         100        I
```

DC1-SPINE2

```
lab@DC1-SPINE2> show route advertising-protocol bgp 10.0.255.3 evpn-esi-
value 05:00:00:fe:4f:00:00:01:2d:00

bgp.evpn.0: 114 destinations, 198 routes (114 active, 0 holddown, 84 hidden)
  Prefix                  Nexthop              MED     Lclpref    AS path
  1:10.0.255.1:0::050000fe4f0000012d00::FFFF:FFFF/304
*                         10.0.255.1                   100        I

default-switch.evpn.0: 40 destinations, 52 routes (40 active, 0 holddown, 12 hidden)

__default_evpn__.evpn.0: 6 destinations, 6 routes (6 active, 0 holddown, 0 hidden)

{master:0}
lab@DC1-SPINE2> show route advertising-protocol bgp 10.0.255.3 evpn-esi-value 05:00:00:fe:
4f:00:00:01:34:00

bgp.evpn.0: 114 destinations, 198 routes (114 active, 0 holddown, 84 hidden)
  Prefix                  Nexthop              MED     Lclpref    AS path
  1:10.0.255.1:0::050000fe4f0000013400::FFFF:FFFF/304
*                         10.0.255.1                   100        I

default-switch.evpn.0: 40 destinations, 52 routes (40 active, 0 holddown, 12 hidden)

__default_evpn__.evpn.0: 6 destinations, 6 routes (6 active, 0 holddown, 0 hidden)
```

Verify Type-1 ESI for T3-1 and T3-8

Now let's jump onto the leaf switches and verify the type-1 EVPN route that was sent by the spine switches for the VGA for each of our virtual networks. One thing to note here is that type-1 EVPN routes for both virtual gateways are advertised to both leaf switches. This is despite DC1-LEAF1 only being interested in VNI301 and DC1-LEAF2 in VNI308. EVPN import policy details exactly what is imported from BGP into the EVPN instance.

DC1-LEAF1

```
lab@DC1-LEAF1> show route evpn-esi-value 05:00:00:fe:4f:00:00:01:2d:00

inet.0: 12 destinations, 12 routes (12 active, 0 holddown, 0 hidden)
```

```
:vxlan.inet.0: 11 destinations, 11 routes (11 active, 0 holddown, 0 hidden)

inet6.0: 1 destinations, 1 routes (1 active, 0 holddown, 0 hidden)

bgp.evpn.0: 27 destinations, 54 routes (14 active, 0 holddown, 26 hidden)
+ = Active Route, - = Last Active, * = Both

1:10.0.255.1:0::050000fe4f0000012d00::FFFF:FFFF/304
                   *[BGP/170] 10w6d 06:40:48, localpref 100, from 10.0.255.1
                      AS path: I, validation-state: unverified
                    > to 172.16.0.14 via xe-0/0/0.0
                    [BGP/170] 10w6d 05:52:06, localpref 100, from 10.0.255.2
                      AS path: I, validation-state: unverified
                    > to 172.16.0.14 via xe-0/0/0.0

default-switch.evpn.0: 29 destinations, 56 routes (16 active, 0 holddown, 26 hidden)
+ = Active Route, - = Last Active, * = Both

1:10.0.255.1:0::050000fe4f0000012d00::FFFF:FFFF/304
                   *[BGP/170] 10w6d 06:40:48, localpref 100, from 10.0.255.1
                      AS path: I, validation-state: unverified
                    > to 172.16.0.14 via xe-0/0/0.0
                    [BGP/170] 10w6d 05:52:06, localpref 100, from 10.0.255.2
                      AS path: I, validation-state: unverified
                    > to 172.16.0.14 via xe-0/0/0.0

{master:0}
lab@DC1-LEAF1> show route evpn-esi-value 05:00:00:fe:4f:00:00:01:34:00

inet.0: 12 destinations, 12 routes (12 active, 0 holddown, 0 hidden)

:vxlan.inet.0: 11 destinations, 11 routes (11 active, 0 holddown, 0 hidden)

inet6.0: 1 destinations, 1 routes (1 active, 0 holddown, 0 hidden)

bgp.evpn.0: 27 destinations, 54 routes (14 active, 0 holddown, 26 hidden)
+ = Active Route, - = Last Active, * = Both

1:10.0.255.1:0::050000fe4f0000013400::FFFF:FFFF/304
                   *[BGP/170] 10w6d 06:54:17, localpref 100, from 10.0.255.1
                      AS path: I, validation-state: unverified
                    > to 172.16.0.14 via xe-0/0/0.0
                    [BGP/170] 10w6d 06:05:35, localpref 100, from 10.0.255.2
                      AS path: I, validation-state: unverified
                    > to 172.16.0.14 via xe-0/0/0.0

default-switch.evpn.0: 29 destinations, 56 routes (16 active, 0 holddown, 26 hidden)
+ = Active Route, - = Last Active, * = Both

1:10.0.255.1:0::050000fe4f0000013400::FFFF:FFFF/304
                   *[BGP/170] 10w6d 06:54:17, localpref 100, from 10.0.255.1
                      AS path: I, validation-state: unverified
                    > to 172.16.0.14 via xe-0/0/0.0
                    [BGP/170] 10w6d 06:05:35, localpref 100, from 10.0.255.2
                      AS path: I, validation-state: unverified
                    > to 172.16.0.14 via xe-0/0/0.0
```

DC1-LEAF2

```
lab@DC1-LEAF2> show route evpn-esi-value 05:00:00:fe:4f:00:00:01:2d:00

inet.0: 12 destinations, 12 routes (12 active, 0 holddown, 0 hidden)

:vxlan.inet.0: 11 destinations, 11 routes (11 active, 0 holddown, 0 hidden)

inet6.0: 1 destinations, 1 routes (1 active, 0 holddown, 0 hidden)

bgp.evpn.0: 27 destinations, 54 routes (14 active, 0 holddown, 26 hidden)
+ = Active Route, - = Last Active, * = Both

1:10.0.255.1:0::050000fe4f0000012d00::FFFF:FFFF/304
                   *[BGP/170] 10w6d 08:18:37, localpref 100, from 10.0.255.1
                      AS path: I, validation-state: unverified
                    > to 172.16.0.16 via xe-0/0/0.0
                    [BGP/170] 10w6d 06:05:02, localpref 100, from 10.0.255.2
                      AS path: I, validation-state: unverified
                    > to 172.16.0.16 via xe-0/0/0.0

default-switch.evpn.0: 29 destinations, 56 routes (16 active, 0 holddown, 26 hidden)
+ = Active Route, - = Last Active, * = Both

1:10.0.255.1:0::050000fe4f0000012d00::FFFF:FFFF/304
                   *[BGP/170] 10w6d 08:18:37, localpref 100, from 10.0.255.1
                      AS path: I, validation-state: unverified
                    > to 172.16.0.16 via xe-0/0/0.0
                    [BGP/170] 10w6d 06:05:02, localpref 100, from 10.0.255.2
                      AS path: I, validation-state: unverified
                    > to 172.16.0.16 via xe-0/0/0.0

{master:0}
lab@DC1-LEAF2> show route evpn-esi-value 05:00:00:fe:4f:00:00:01:34:00

inet.0: 12 destinations, 12 routes (12 active, 0 holddown, 0 hidden)

:vxlan.inet.0: 11 destinations, 11 routes (11 active, 0 holddown, 0 hidden)

inet6.0: 1 destinations, 1 routes (1 active, 0 holddown, 0 hidden)

bgp.evpn.0: 27 destinations, 54 routes (14 active, 0 holddown, 26 hidden)
+ = Active Route, - = Last Active, * = Both

1:10.0.255.1:0::050000fe4f0000013400::FFFF:FFFF/304
                   *[BGP/170] 10w6d 08:21:13, localpref 100, from 10.0.255.1
                      AS path: I, validation-state: unverified
                    > to 172.16.0.16 via xe-0/0/0.0
                    [BGP/170] 10w6d 06:07:38, localpref 100, from 10.0.255.2
                      AS path: I, validation-state: unverified
                    > to 172.16.0.16 via xe-0/0/0.0

default-switch.evpn.0: 29 destinations, 56 routes (16 active, 0 holddown, 26 hidden)
+ = Active Route, - = Last Active, * = Both

1:10.0.255.1:0::050000fe4f0000013400::FFFF:FFFF/304
                   *[BGP/170] 10w6d 08:21:13, localpref 100, from 10.0.255.1
```

```
                    AS path: I, validation-state: unverified
                  > to 172.16.0.16 via xe-0/0/0.0
                  [BGP/170] 10w6d 06:07:38, localpref 100, from 10.0.255.2
                    AS path: I, validation-state: unverified
                  > to 172.16.0.16 via xe-0/0/0.0
```

Verify EVPN Database on Leaf Switches

Now let's check out the EVPN database on the leaf switches. Here we can see the Virtual Gateway MAC address and IP address. You can also see the ESI (active source) for the Virtual Gateway segment.

DC1-LEAF1

```
lab@DC1-LEAF1> show evpn database
Instance: default-switch
VLAN  VNI  MAC address        Active source                  Timestamp        IP address
      301   00:00:5e:00:01:01  05:00:00:fe:50:00:00:01:2d:00  Mar 19 01:34:09  192.168.31.254
      301   00:0c:29:f4:41:bb  xe-0/0/3.0                     May 15 07:31:03
      301   02:05:86:71:03:00  10.0.255.2                     May 22 14:41:20  192.168.31.253
      301   02:05:86:71:d1:00  10.0.255.1                     Mar 19 00:45:31  192.168.31.252
```

DC1-LEAF2

```
lab@DC1-LEAF2> show evpn database
Instance: default-switch
VLAN  VNI  MAC address        Active source                  Timestamp        IP address
      308   00:00:5e:00:01:01  05:00:00:fe:50:00:00:01:34:00  Mar 19 01:34:13  192.168.38.254
      308   00:0c:29:8e:e8:6d  xe-0/0/3.0                     May 15 07:38:00
      308   02:05:86:71:03:00  10.0.255.2                     May 22 14:41:20  192.168.38.253
      308   02:05:86:71:d1:00  10.0.255.1                     Mar 18 23:22:49  192.168.38.252
```

Verify Tenant Routing Table on Spine Switches

Next let's jump back to the spine switches to verify the tenant VRF has both tenant subnets reachable locally via the IRB interfaces.

DC1-SPINE1

```
lab@DC1-SPINE1> show route table TENANT_3_VRF.inet.0

TENANT_3_VRF.inet.0: 5 destinations, 5 routes (5 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.30.255.1/32     *[Direct/0] 31w3d 16:14:55
                    > via lo0.30
192.168.31.0/24    *[Direct/0] 1w4d 18:08:05
                    > via irb.301
192.168.31.252/32  *[Local/0] 1w4d 18:08:05
                      Local via irb.301
192.168.38.0/24    *[Direct/0] 1w4d 18:08:05
                    > via irb.308
192.168.38.252/32  *[Local/0] 1w4d 18:08:05
                      Local via irb.308
```

DC1-SPINE2

```
lab@DC1-SPINE2> show route table TENANT_3_VRF.inet.0

TENANT_3_VRF.inet.0: 5 destinations, 5 routes (5 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.30.255.2/32     *[Direct/0] 31w3d 16:26:15
                    > via lo0.30
192.168.31.0/24    *[Direct/0] 1w4d 18:19:03
                    > via irb.301
192.168.31.253/32  *[Local/0] 1w4d 18:19:03
                      Local via irb.301
192.168.38.0/24    *[Direct/0] 1w4d 18:19:03
                    > via irb.308
192.168.38.253/32  *[Local/0] 1w4d 18:19:03
                      Local via irb.308
```

Send IP Flow Between Hosts

Now that we've completed the verification, let's send an IP flow between T3-1 and T3-8.

T3-1

```
root@t3-1:~# ping 192.168.38.1 -c 5
PING 192.168.38.1 (192.168.38.1) 56(84) bytes of data.
64 bytes from 192.168.38.1: icmp_seq=1 ttl=63 time=122 ms
64 bytes from 192.168.38.1: icmp_seq=2 ttl=63 time=32.9 ms
64 bytes from 192.168.38.1: icmp_seq=3 ttl=63 time=832 ms
64 bytes from 192.168.38.1: icmp_seq=4 ttl=63 time=445 ms
64 bytes from 192.168.38.1: icmp_seq=5 ttl=63 time=35.0 ms
```

Capture ARP Request for IP Flow Between Tenant Hosts

Lastly, we'll capture the traffic for the above flow at host T3-8. Here you can see an ARP request is sent from T3-8 for 192.168.38.254. A response is received from both spine switches, but, crucially, they respond with the same anycast gateway MAC 00:00:5e:00:01:01.

T3-8

```
12:14:53.126066 ARP, Ethernet (len 6), IPv4 (len 4), Request who-
has 192.168.38.254 tell 192.168.38.1, length 28
12:14:53.155379 ARP, Ethernet (len 6), IPv4 (len 4), Reply 192.168.38.254 is-
at 00:00:5e:00:01:01 (oui IANA), length 46
12:14:53.348585 ARP, Ethernet (len 6), IPv4 (len 4), Reply 192.168.38.254 is-
at 00:00:5e:00:01:01 (oui IANA), length 46
```

Discussion

EVPN anycast gateway is an extremely useful feature that can be used to provide a L3 gateway for virtual networks. It also provides a mechanism for making the gateway redundant. Typically anycast gateway is deployed at the spine layer in an IP fabric, but as switch hardware evolves you may start to see the feature deployed at the leaf.

Recipe 9: Configuring Redundant DCI Using EVPN Route Type-5 on Juniper QFX10K

By Dan Hearty

- Junos OS Used: 15.1X53-D60.4
- Juniper Platforms General Applicability: QFX Series Switches

EVPN route type-5 provides a mechanism for aggregating multiple host MAC+IP routes for tenants behind a single IP prefix for a given bridge domain. This can be very useful with regard to Data Center Interconnect (DCI), whereby MAC+IP routes no longer have to be sent over DCIs and a single IP prefix route can be used instead.

Problem

In this scenario there are two data centers, DC1 and DC2, with two DCI circuits connected between two pairs of spine switches. Hosts in each DC utilize EVPN virtual gateway to reach remote subnets. Hosts may use either spine switch as a gateway, thus each spine switch *must maintain connectivity to the remote DC, even in the event of a single DCI failure*. There is no requirement to provide stretched Layer 2 between the data centers, so host MAC+IP routes should be aggregated behind a single IP prefix.

Solution

To solve this problem, EVPN route type-5 will be used to aggregate host MAC+IP routes behind a single IP prefix for each given bridge domain. EVPN type-5 routes will be exchanged over both data center interconnects and between spine switches within a given data center. Figures 9.1 through 9.3 illustrate the data center fabrics and DCIs used in this recipe.

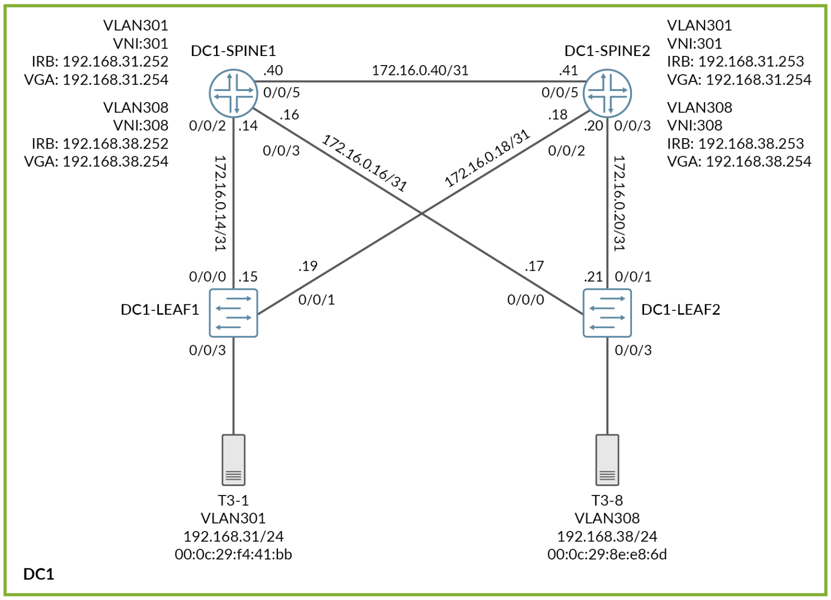


Figure 9.1 *DC1 IP Fabric*

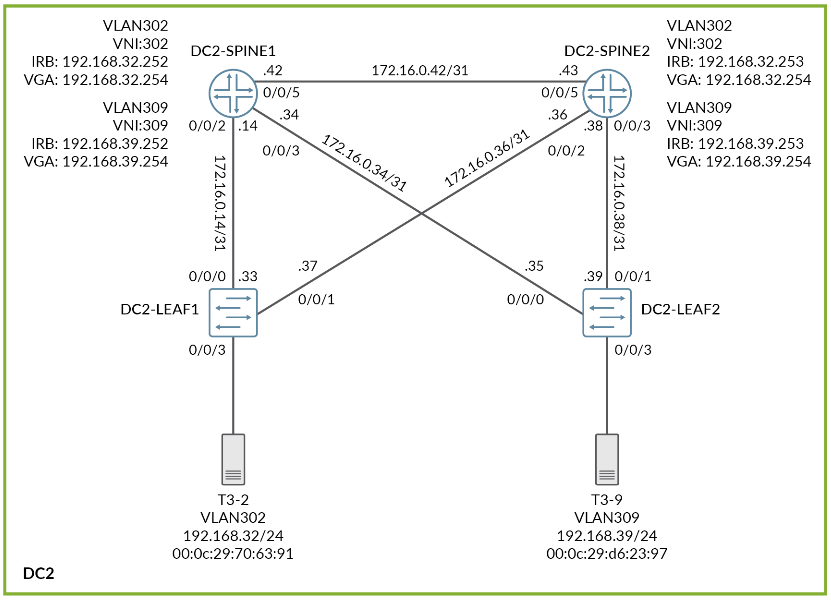


Figure 9.2 *DC2 IP Fabric*

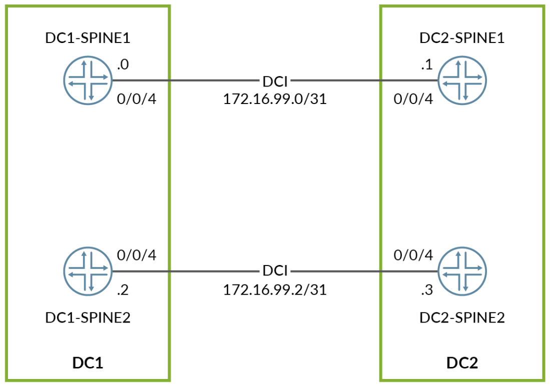


Figure 9.3 Data Center Interconnect

Each data center network is based on an EVPN-VXLAN IP fabric. BGP is used in both the underlay network and the overlay network, as well as on the data center interconnects.

Data Center Underlay Network

DC1 and DC2 are both based on an ECMP IP fabric utilizing EBGP. The architecture of the data centers is identical with the exception of IP addressing, ASN assignments, and various EVPN variables.

The configuration statements for the underlay network used within each data center have been omitted, as they bear little relevance to the EVPN type-5 solution. However, the configuration statements required to enable the underlay network for the data center interconnects, which are also based on EBGP, are included.

Data Center Interconnect Underlay Network

This section details the configuration steps for enabling the DCI underlay network. The objective of the DCI underlay network is to exchange loopback information with the remotely connected spine switch.

The following section details the configuration steps to enable the DCI underlay for both DC1 and DC2 spine switches.

Configure DCI EBGP Underlay

Here we configure an EBGP peering on the physical DCI link between spine switches for each given data center interconnect.

DC1-SPINE1

```
lab@DC1-SPINE1> show configuration protocols bgp group dci-underlay
type external;
mtu-discovery;
import bgp-ipclos-in;
export bgp-dci-out;
local-as 65103;
neighbor 172.16.99.1 {
    peer-as 65203;
}

lab@DC1-SPINE1> show configuration protocols bgp group dci-underlay | display set
set protocols bgp group dci-underlay type external
set protocols bgp group dci-underlay mtu-discovery
set protocols bgp group dci-underlay import bgp-ipclos-in
set protocols bgp group dci-underlay export bgp-dci-out
set protocols bgp group dci-underlay local-as 65103
set protocols bgp group dci-underlay neighbor 172.16.99.1 peer-as 65203
```

DC2-SPINE1

```
lab@DC2-SPINE1> show configuration protocols bgp group dci-underlay
type external;
mtu-discovery;
import bgp-ipclos-in;
export bgp-dci-out;
local-as 65203;
neighbor 172.16.99.0 {
    peer-as 65103;
}

lab@DC2-SPINE1> show configuration protocols bgp group dci-underlay | display set
set protocols bgp group dci-underlay type external
set protocols bgp group dci-underlay mtu-discovery
set protocols bgp group dci-underlay import bgp-ipclos-in
set protocols bgp group dci-underlay export bgp-dci-out
set protocols bgp group dci-underlay local-as 65203
set protocols bgp group dci-underlay neighbor 172.16.99.0 peer-as 65103
```

DC1-SPINE2

```
lab@DC1-SPINE2> show configuration protocols bgp group dci-underlay
type external;
mtu-discovery;
import bgp-ipclos-in;
export bgp-dci-out;
local-as 65104;
neighbor 172.16.99.3 {
    peer-as 65204;
}

lab@DC1-SPINE2> show configuration protocols bgp group dci-underlay | display set
set protocols bgp group dci-underlay type external
set protocols bgp group dci-underlay mtu-discovery
set protocols bgp group dci-underlay import bgp-ipclos-in
set protocols bgp group dci-underlay export bgp-dci-out
set protocols bgp group dci-underlay local-as 65104
set protocols bgp group dci-underlay neighbor 172.16.99.3 peer-as 65204
```

DC2-SPINE2

```
lab@DC2-SPINE2> show configuration protocols bgp group dci-underlay
type external;
mtu-discovery;
import bgp-ipclos-in;
export bgp-dci-out;
local-as 65204;
neighbor 172.16.99.2 {
    peer-as 65104;
}

lab@DC2-SPINE2> show configuration protocols bgp group dci-underlay | display set
set protocols bgp group dci-underlay type external
set protocols bgp group dci-underlay mtu-discovery
set protocols bgp group dci-underlay import bgp-ipclos-in
set protocols bgp group dci-underlay export bgp-dci-out
set protocols bgp group dci-underlay local-as 65204
set protocols bgp group dci-underlay neighbor 172.16.99.2 peer-as 65104
```

Configure DCI EBGP Underlay Import and Export Policy

In the previous step we defined an import and export policy for the underlay DCI EBGP peering. The export policy is used to announce the local /32 loopback and the remote spine /32 loopback into BGP. The import policy is used to define what should be imported; in this example it's the subnet that is used for loopback interfaces.

DC1-SPINE1

Export Policy

```
lab@DC1-SPINE1> show configuration policy-options policy-statement bgp-ipclos-out
term loopback {
    from {
        family inet;
        protocol direct;
        route-filter 0.0.0.0/0 prefix-length-range /32-/32;
    }
    then {
        community add MYCOMMUNITY;
        next-hop self;
        accept;
    }
}
term remote-loopback {
    from {
        protocol bgp;
        route-filter 10.0.255.5/32 exact;
    }
    then accept;
}
term reject {
    then reject;
}
```

```
lab@DC1-SPINE1> show configuration policy-options policy-statement bgp-ipclos-out | display set
set policy-options policy-statement bgp-ipclos-out term loopback from family inet
set policy-options policy-statement bgp-ipclos-out term loopback from protocol direct
set policy-options policy-statement bgp-ipclos-out term loopback from route-filter 0.0.0.0/0 prefix-
length-range /32-/32
set policy-options policy-statement bgp-ipclos-out term loopback then community add MYCOMMUNITY
set policy-options policy-statement bgp-ipclos-out term loopback then next-hop self
set policy-options policy-statement bgp-ipclos-out term loopback then accept
set policy-options policy-statement bgp-ipclos-out term remote-loopback from protocol bgp
set policy-options policy-statement bgp-ipclos-out term remote-loopback from route-
filter 10.0.255.5/32 exact
set policy-options policy-statement bgp-ipclos-out term remote-loopback then accept
set policy-options policy-statement bgp-ipclos-out term reject then reject
```

Import Policy

```
lab@DC1-SPINE1> show configuration policy-options policy-statement bgp-ipclos-in
term loopbacks {
    from {
        route-filter 10.0.255.0/24 orlonger;
    }
    then accept;
}
term reject {
    then reject;
}

lab@DC1-SPINE1> show configuration policy-options policy-statement bgp-ipclos-in | display set
set policy-options policy-statement bgp-ipclos-in term loopbacks from route-
filter 10.0.255.0/24 orlonger
set policy-options policy-statement bgp-ipclos-in term loopbacks then accept
set policy-options policy-statement bgp-ipclos-in term reject then reject
```

DC2-SPINE1

Export Policy

```
lab@DC2-SPINE1> show configuration policy-options policy-statement bgp-ipclos-out
term loopback {
    from {
        family inet;
        protocol direct;
        route-filter 0.0.0.0/0 prefix-length-range /32-/32;
    }
    then {
        community add MYCOMMUNITY;
        next-hop self;
        accept;
    }
}
term remote-loopback {
    from {
        protocol bgp;
        route-filter 10.0.255.1/32 exact;
    }
    then accept;
}
term reject {
    then reject;
}
```

```
lab@DC2-SPINE1> show configuration policy-options policy-statement bgp-ipclos-out | display set
set policy-options policy-statement bgp-ipclos-out term loopback from family inet
set policy-options policy-statement bgp-ipclos-out term loopback from protocol direct
set policy-options policy-statement bgp-ipclos-out term loopback from route-filter 0.0.0.0/0 prefix-
length-range /32-/32
set policy-options policy-statement bgp-ipclos-out term loopback then community add MYCOMMUNITY
set policy-options policy-statement bgp-ipclos-out term loopback then next-hop self
set policy-options policy-statement bgp-ipclos-out term loopback then accept
set policy-options policy-statement bgp-ipclos-out term remote-loopback from protocol bgp
set policy-options policy-statement bgp-ipclos-out term remote-loopback from route-
filter 10.0.255.1/32 exact
set policy-options policy-statement bgp-ipclos-out term remote-loopback then accept
set policy-options policy-statement bgp-ipclos-out term reject then reject
```

Import Policy

```
lab@DC2-SPINE1> show configuration policy-options policy-statement bgp-ipclos-in
term loopbacks {
    from {
        route-filter 10.0.255.0/24 orlonger;
    }
    then accept;
}
term reject {
    then reject;
}

lab@DC2-SPINE1> show configuration policy-options policy-statement bgp-ipclos-in | display set
set policy-options policy-statement bgp-ipclos-in term loopbacks from route-
filter 10.0.255.0/24 orlonger
set policy-options policy-statement bgp-ipclos-in term loopbacks then accept
set policy-options policy-statement bgp-ipclos-in term reject then reject
```

DC1-SPINE2

Export Policy

```
lab@DC1-SPINE2> show configuration policy-options policy-statement bgp-ipclos-out
term loopback {
    from {
        family inet;
        protocol direct;
        route-filter 0.0.0.0/0 prefix-length-range /32-/32;
    }
    then {
        community add MYCOMMUNITY;
        next-hop self;
        accept;
    }
}
term remote-loopback {
    from {
        protocol bgp;
        route-filter 10.0.255.6/32 exact;
    }
    then accept;
}
term reject {
    then reject;
}
```

```
lab@DC1-SPINE2> show configuration policy-options policy-statement bgp-ipclos-out | display set
set policy-options policy-statement bgp-ipclos-out term loopback from family inet
set policy-options policy-statement bgp-ipclos-out term loopback from protocol direct
set policy-options policy-statement bgp-ipclos-out term loopback from route-filter 0.0.0.0/0 prefix-
length-range /32-/32
set policy-options policy-statement bgp-ipclos-out term loopback then community add MYCOMMUNITY
set policy-options policy-statement bgp-ipclos-out term loopback then next-hop self
set policy-options policy-statement bgp-ipclos-out term loopback then accept
set policy-options policy-statement bgp-ipclos-out term remote-loopback from protocol bgp
set policy-options policy-statement bgp-ipclos-out term remote-loopback from route-
filter 10.0.255.6/32 exact
set policy-options policy-statement bgp-ipclos-out term remote-loopback then accept
set policy-options policy-statement bgp-ipclos-out term reject then reject
```

Import Policy

```
lab@DC1-SPINE2> show configuration policy-options policy-statement bgp-ipclos-in
term loopbacks {
    from {
        route-filter 10.0.255.0/24 orlonger;
    }
    then accept;
}
term reject {
    then reject;
}

lab@DC1-SPINE2> show configuration policy-options policy-statement bgp-ipclos-in | display set
set policy-options policy-statement bgp-ipclos-in term loopbacks from route-
filter 10.0.255.0/24 orlonger
set policy-options policy-statement bgp-ipclos-in term loopbacks then accept
set policy-options policy-statement bgp-ipclos-in term reject then reject
```

DC2-SPINE2

Export Policy

```
lab@DC2-SPINE2> show configuration policy-options policy-statement bgp-ipclos-out
term loopback {
    from {
        family inet;
        protocol direct;
        route-filter 0.0.0.0/0 prefix-length-range /32-/32;
    }
    then {
        community add MYCOMMUNITY;
        next-hop self;
        accept;
    }
}
term remote-loopback {
    from {
        protocol bgp;
        route-filter 10.0.255.2/32 exact;
    }
    then accept;
}
term reject {
    then reject;
}
```

```
lab@DC2-SPINE2> show configuration policy-options policy-statement bgp-ipclos-out | display set
set policy-options policy-statement bgp-ipclos-out term loopback from family inet
set policy-options policy-statement bgp-ipclos-out term loopback from protocol direct
set policy-options policy-statement bgp-ipclos-out term loopback from route-filter 0.0.0.0/0 prefix-
length-range /32-/32
set policy-options policy-statement bgp-ipclos-out term loopback then community add MYCOMMUNITY
set policy-options policy-statement bgp-ipclos-out term loopback then next-hop self
set policy-options policy-statement bgp-ipclos-out term loopback then accept
set policy-options policy-statement bgp-ipclos-out term remote-loopback from protocol bgp
set policy-options policy-statement bgp-ipclos-out term remote-loopback from route-
filter 10.0.255.2/32 exact
set policy-options policy-statement bgp-ipclos-out term remote-loopback then accept
set policy-options policy-statement bgp-ipclos-out term reject then reject
```

Import Policy

```
lab@DC2-SPINE2> show configuration policy-options policy-statement bgp-ipclos-in
term loopbacks {
    from {
        route-filter 10.0.255.0/24 orlonger;
    }
    then accept;
}
term reject {
    then reject;
}

lab@DC2-SPINE2> show configuration policy-options policy-statement bgp-ipclos-in | display set
set policy-options policy-statement bgp-ipclos-in term loopbacks from route-
filter 10.0.255.0/24 orlonger
set policy-options policy-statement bgp-ipclos-in term loopbacks then accept
set policy-options policy-statement bgp-ipclos-in term reject then reject
```

Data Center Overlay Network

The overlay network for DC1 and DC2 is based on EVPN with VXLAN overlay tunnels. IBGP is used in each data center for signalling EVPN in a route-reflector architecture, whereby the spine switches act as BGP route reflectors and the leaf switches act as BGP route-reflector clients in each given data center.

The configuration steps required to enable the overlay network within each DC matches that of *Recipe 8: Configuring EVPN Anycast Gateway with Intra-Tenant Inter-Subnet Routing*. Refer to that recipe for a detailed description of each step.

Configure Data Center Overlay Network

The following section details the configuration steps to enable the overlay network in DC1 and DC2.

DC1-SPINE1

BGP RR Overlay

```
lab@DC1-SPINE1> show configuration protocols bgp group overlay-evpn-rr
type internal;
```

```
local-address 10.0.255.1;
family evpn {
    signaling;
}
vpn-apply-export;
cluster 1.1.1.1;
local-as 65001;
multipath;
neighbor 10.0.255.3;
neighbor 10.0.255.4;

lab@DC1-SPINE1> show configuration protocols bgp group overlay-evpn-rr | display set
set protocols bgp group overlay-evpn-rr type internal
set protocols bgp group overlay-evpn-rr local-address 10.0.255.1
set protocols bgp group overlay-evpn-rr family evpn signaling
set protocols bgp group overlay-evpn-rr vpn-apply-export
set protocols bgp group overlay-evpn-rr cluster 1.1.1.1
set protocols bgp group overlay-evpn-rr local-as 65001
set protocols bgp group overlay-evpn-rr multipath
set protocols bgp group overlay-evpn-rr neighbor 10.0.255.3
set protocols bgp group overlay-evpn-rr neighbor 10.0.255.4
```

BGP Overlay

```
lab@DC1-SPINE1> show configuration protocols bgp group overlay-evpn
type internal;
local-address 10.0.255.1;
family evpn {
    signaling;
}
local-as 65001;
multipath;
neighbor 10.0.255.2;

lab@DC1-SPINE1> show configuration protocols bgp group overlay-evpn | display set
set protocols bgp group overlay-evpn type internal
set protocols bgp group overlay-evpn local-address 10.0.255.1
set protocols bgp group overlay-evpn family evpn signaling
set protocols bgp group overlay-evpn local-as 65001
set protocols bgp group overlay-evpn multipath
set protocols bgp group overlay-evpn neighbor 10.0.255.2
```

EVPN

```
lab@DC1-SPINE1> show configuration protocols evpn
encapsulation vxlan;
multicast-mode ingress-replication;
default-gateway no-gateway-community;

lab@DC1-SPINE1> show configuration protocols evpn | display set
set protocols evpn encapsulation vxlan
set protocols evpn multicast-mode ingress-replication
set protocols evpn default-gateway no-gateway-community
```

Switch Options

```
lab@DC1-SPINE1> show configuration switch-options
vtep-source-interface lo0.0;
```

```
route-distinguisher 10.0.255.1:1;
vrf-import EVPN_IMPORT;
vrf-target target:9999:9999;

lab@DC1-SPINE1> show configuration switch-options | display set
set switch-options vtep-source-interface lo0.0
set switch-options route-distinguisher 10.0.255.1:1
set switch-options vrf-import EVPN_IMPORT
set switch-options vrf-target target:9999:9999
```

EVPN Import Policy

```
lab@DC1-SPINE1> show configuration policy-options policy-statement EVPN_IMPORT
term import_ESI {
    from community ESI;
    then accept;
}
term last {
    then reject;
}

lab@DC1-SPINE1> show configuration policy-options policy-statement EVPN_IMPORT | display set
set policy-options policy-statement EVPN_IMPORT term import_ESI from community ESI
set policy-options policy-statement EVPN_IMPORT term import_ESI then accept
set policy-options policy-statement EVPN_IMPORT term last then reject
```

ESI Community

```
lab@DC1-SPINE1> show configuration policy-options community ESI
members target:9999:9999;

lab@DC1-SPINE1> show configuration policy-options community ESI | display set
set policy-options community ESI members target:9999:9999
```

DC1-SPINE2

BGP RR Overlay

```
lab@DC1-SPINE2> show configuration protocols bgp group overlay-evpn-rr
type internal;
local-address 10.0.255.2;
family evpn {
    signaling;
}
vpn-apply-export;
cluster 1.1.1.1;
local-as 65001;
multipath;
neighbor 10.0.255.3;
neighbor 10.0.255.4;

lab@DC1-SPINE2> show configuration protocols bgp group overlay-evpn-rr | display set
set protocols bgp group overlay-evpn-rr type internal
set protocols bgp group overlay-evpn-rr local-address 10.0.255.2
set protocols bgp group overlay-evpn-rr family evpn signaling
set protocols bgp group overlay-evpn-rr vpn-apply-export
set protocols bgp group overlay-evpn-rr cluster 1.1.1.1
set protocols bgp group overlay-evpn-rr local-as 65001
set protocols bgp group overlay-evpn-rr multipath
set protocols bgp group overlay-evpn-rr neighbor 10.0.255.3
set protocols bgp group overlay-evpn-rr neighbor 10.0.255.4
```

BGP Overlay

```
lab@DC1-SPINE2> show configuration protocols bgp group overlay-evpn
type internal;
local-address 10.0.255.2;
family evpn {
    signaling;
}
local-as 65001;
multipath;
neighbor 10.0.255.1;

lab@DC1-SPINE2> show configuration protocols bgp group overlay-evpn | display set
set protocols bgp group overlay-evpn type internal
set protocols bgp group overlay-evpn local-address 10.0.255.2
set protocols bgp group overlay-evpn family evpn signaling
set protocols bgp group overlay-evpn local-as 65001
set protocols bgp group overlay-evpn multipath
set protocols bgp group overlay-evpn neighbor 10.0.255.1
```

EVPN

```
lab@DC1-SPINE2> show configuration protocols evpn
encapsulation vxlan;
multicast-mode ingress-replication;
default-gateway no-gateway-community;

lab@DC1-SPINE2> show configuration protocols evpn | display set
set protocols evpn encapsulation vxlan
set protocols evpn multicast-mode ingress-replication
set protocols evpn default-gateway no-gateway-community
```

Switch Options

```
lab@DC1-SPINE2> show configuration switch-options
vtep-source-interface lo0.0;
route-distinguisher 10.0.255.2:1;
vrf-import EVPN_IMPORT;
vrf-target target:9999:9999;

lab@DC1-SPINE2> show configuration switch-options | display set
set switch-options vtep-source-interface lo0.0
set switch-options route-distinguisher 10.0.255.2:1
set switch-options vrf-import EVPN_IMPORT
set switch-options vrf-target target:9999:9999
```

EVPN Import Policy

```
lab@DC1-SPINE2> show configuration policy-options policy-statement EVPN_IMPORT
term import_ESI {
    from community ESI;
    then accept;
}
```

```
term last {
    then reject;
}

lab@DC1-SPINE2> show configuration policy-options policy-statement EVPN_IMPORT | display set
set policy-options policy-statement EVPN_IMPORT term import_ESI from community ESI
set policy-options policy-statement EVPN_IMPORT term import_ESI then accept
set policy-options policy-statement EVPN_IMPORT term last then reject
```

ESI Community

```
lab@DC1-SPINE2> show configuration policy-options community ESI
members target:9999:9999;

lab@DC1-SPINE2> show configuration policy-options community ESI | display set
set policy-options community ESI members target:9999:9999
```

DC1-LEAF1

BGP Overlay

```
lab@DC1-LEAF1> show configuration protocols bgp group overlay-evpn
type internal;
local-address 10.0.255.3;
family evpn {
    signaling;
}
local-as 65001;
multipath;
neighbor 10.0.255.1;
neighbor 10.0.255.2;

lab@DC1-LEAF1> show configuration protocols bgp group overlay-evpn | display set
set protocols bgp group overlay-evpn type internal
set protocols bgp group overlay-evpn local-address 10.0.255.3
set protocols bgp group overlay-evpn family evpn signaling
set protocols bgp group overlay-evpn local-as 65001
set protocols bgp group overlay-evpn multipath
set protocols bgp group overlay-evpn neighbor 10.0.255.1
set protocols bgp group overlay-evpn neighbor 10.0.255.2
```

EVPN

```
lab@DC1-LEAF1> show configuration protocols evpn
encapsulation vxlan;
multicast-mode ingress-replication;

lab@DC1-LEAF1> show configuration protocols evpn | display set
set protocols evpn encapsulation vxlan
set protocols evpn multicast-mode ingress-replication
```

Switch-Options

```
lab@DC1-LEAF1> show configuration switch-options
vtep-source-interface lo0.0;
route-distinguisher 10.0.255.3:1;
vrf-import EVPN_IMPORT;
vrf-target target:9999:9999;
```

```
lab@DC1-LEAF1> show configuration switch-options | display set
set switch-options vtep-source-interface lo0.0
set switch-options route-distinguisher 10.0.255.3:1
set switch-options vrf-import EVPN_IMPORT
set switch-options vrf-target target:9999:9999
```

EVPN Import Policy

```
lab@DC1-LEAF1> show configuration policy-options policy-statement EVPN_IMPORT
term import_ESI {
    from community ESI;
    then accept;
}
term last {
    then reject;
}

lab@DC1-LEAF1> show configuration policy-options policy-statement EVPN_IMPORT | display set
set policy-options policy-statement EVPN_IMPORT term import_ESI from community ESI
set policy-options policy-statement EVPN_IMPORT term import_ESI then accept
set policy-options policy-statement EVPN_IMPORT term last then reject
```

ESI Community

```
lab@DC1-LEAF1> show configuration policy-options community ESI
members target:9999:9999;

lab@DC1-LEAF1> show configuration policy-options community ESI | display set
set policy-options community ESI members target:9999:9999
```

DC1-LEAF2

BGP Overlay

```
lab@DC1-LEAF2> show configuration protocols bgp group overlay-evpn
type internal;
local-address 10.0.255.4;
family evpn {
    signaling;
}
local-as 65001;
multipath;
neighbor 10.0.255.1;
neighbor 10.0.255.2;

lab@DC1-LEAF2> show configuration protocols bgp group overlay-evpn | display set
set protocols bgp group overlay-evpn type internal
set protocols bgp group overlay-evpn local-address 10.0.255.4
set protocols bgp group overlay-evpn family evpn signaling
set protocols bgp group overlay-evpn local-as 65001
set protocols bgp group overlay-evpn multipath
set protocols bgp group overlay-evpn neighbor 10.0.255.1
set protocols bgp group overlay-evpn neighbor 10.0.255.2
```

EVPN

```
lab@DC1-LEAF2> show configuration protocols evpn
encapsulation vxlan;
multicast-mode ingress-replication;
```

```
lab@DC1-LEAF2> show configuration protocols evpn | display set
set protocols evpn encapsulation vxlan
set protocols evpn multicast-mode ingress-replication
```

Switch Options

```
lab@DC1-LEAF2> show configuration switch-options
vtep-source-interface lo0.0;
route-distinguisher 10.0.255.4:1;
vrf-import EVPN_IMPORT;
vrf-target target:9999:9999;

lab@DC1-LEAF2> show configuration switch-options | display set
set switch-options vtep-source-interface lo0.0
set switch-options route-distinguisher 10.0.255.4:1
set switch-options vrf-import EVPN_IMPORT
set switch-options vrf-target target:9999:9999
```

EVPN Import Policy

```
lab@DC1-LEAF2> show configuration policy-options policy-statement EVPN_IMPORT
term import_ESI {
    from community ESI;
    then accept;
}
term last {
    then reject;
}

lab@DC1-LEAF2> show configuration policy-options policy-statement EVPN_IMPORT | display set
set policy-options policy-statement EVPN_IMPORT term import_ESI from community ESI
set policy-options policy-statement EVPN_IMPORT term import_ESI then accept
set policy-options policy-statement EVPN_IMPORT term last then reject
```

ESI Community

```
lab@DC1-LEAF2> show configuration policy-options community ESI
members target:9999:9999;

lab@DC1-LEAF2> show configuration policy-options community ESI | display set
set policy-options community ESI members target:9999:9999
```

DC2-SPINE1

BGP RR Overlay

```
lab@DC2-SPINE1> show configuration protocols bgp group overlay-evpn-rr
type internal;
local-address 10.0.255.5;
family evpn {
    signaling;
}
vpn-apply-export;
cluster 2.2.2.2;
local-as 65002;
multipath;
neighbor 10.0.255.7;
neighbor 10.0.255.8;

lab@DC2-SPINE1> show configuration protocols bgp group overlay-evpn-rr | display set
set protocols bgp group overlay-evpn-rr type internal
set protocols bgp group overlay-evpn-rr local-address 10.0.255.5
set protocols bgp group overlay-evpn-rr family evpn signaling
set protocols bgp group overlay-evpn-rr vpn-apply-export
set protocols bgp group overlay-evpn-rr cluster 2.2.2.2
set protocols bgp group overlay-evpn-rr local-as 65002
set protocols bgp group overlay-evpn-rr multipath
set protocols bgp group overlay-evpn-rr neighbor 10.0.255.7
set protocols bgp group overlay-evpn-rr neighbor 10.0.255.8
```

BGP Overlay

```
lab@DC2-SPINE1> show configuration protocols bgp group overlay-evpn
type internal;
local-address 10.0.255.5;
family evpn {
    signaling;
}
local-as 65002;
multipath;
neighbor 10.0.255.6;

lab@DC2-SPINE1> show configuration protocols bgp group overlay-evpn | display set
set protocols bgp group overlay-evpn type internal
set protocols bgp group overlay-evpn local-address 10.0.255.5
set protocols bgp group overlay-evpn family evpn signaling
set protocols bgp group overlay-evpn local-as 65002
set protocols bgp group overlay-evpn multipath
set protocols bgp group overlay-evpn neighbor 10.0.255.6
```

EVPN

```
lab@DC2-SPINE1> show configuration protocols evpn
encapsulation vxlan;
multicast-mode ingress-replication;

lab@DC2-SPINE1> show configuration protocols evpn | display set
set protocols evpn encapsulation vxlan
set protocols evpn multicast-mode ingress-replication
```

Switch Options

```
lab@DC2-SPINE1> show configuration switch-options
vtep-source-interface lo0.0;
route-distinguisher 10.0.255.5:1;
vrf-import EVPN_IMPORT;
vrf-target target:9999:9999;

lab@DC2-SPINE1> show configuration switch-options | display set
set switch-options vtep-source-interface lo0.0
set switch-options route-distinguisher 10.0.255.5:1
set switch-options vrf-import EVPN_IMPORT
set switch-options vrf-target target:9999:9999
```

EVPN Import Policy

```
lab@DC2-SPINE1> show configuration policy-options policy-statement EVPN_IMPORT
term import_ESI {
    from community ESI;
    then accept;
}
term last {
    then reject;
}

lab@DC2-SPINE1> show configuration policy-options policy-statement EVPN_IMPORT | display set
set policy-options policy-statement EVPN_IMPORT term import_ESI from community ESI
set policy-options policy-statement EVPN_IMPORT term import_ESI then accept
set policy-options policy-statement EVPN_IMPORT term last then reject
```

ESI Community

```
lab@DC2-SPINE1> show configuration policy-options community ESI
members target:9999:9999;

lab@DC2-SPINE1> show configuration policy-options community ESI | display set
set policy-options community ESI members target:9999:9999
```

DC2-SPINE2

BGP RR Overlay

```
lab@DC2-SPINE2> show configuration protocols bgp group overlay-evpn-rr
type internal;
local-address 10.0.255.6;
family evpn {
    signaling;
}
vpn-apply-export;
cluster 2.2.2.2;
local-as 65002;
multipath;
neighbor 10.0.255.7;
neighbor 10.0.255.8;

lab@DC2-SPINE2> show configuration protocols bgp group overlay-evpn-rr | display set
set protocols bgp group overlay-evpn-rr type internal
set protocols bgp group overlay-evpn-rr local-address 10.0.255.6
set protocols bgp group overlay-evpn-rr family evpn signaling
set protocols bgp group overlay-evpn-rr vpn-apply-export
set protocols bgp group overlay-evpn-rr cluster 2.2.2.2
set protocols bgp group overlay-evpn-rr local-as 65002
set protocols bgp group overlay-evpn-rr multipath
set protocols bgp group overlay-evpn-rr neighbor 10.0.255.7
set protocols bgp group overlay-evpn-rr neighbor 10.0.255.8
```

BGP Overlay

```
lab@DC2-SPINE2> show configuration protocols bgp group overlay-evpn
show configuration protocols bgp group overlay-evpn | display set type internal;
local-address 10.0.255.6;
family evpn {
    signaling;
}
local-as 65002;
multipath;
neighbor 10.0.255.5;

lab@DC2-SPINE2> show configuration protocols bgp group overlay-evpn | display set
set protocols bgp group overlay-evpn type internal
set protocols bgp group overlay-evpn local-address 10.0.255.6
set protocols bgp group overlay-evpn family evpn signaling
set protocols bgp group overlay-evpn local-as 65002
set protocols bgp group overlay-evpn multipath
set protocols bgp group overlay-evpn neighbor 10.0.255.5
```

EVPN

```
lab@DC2-SPINE2> show configuration protocols evpn
encapsulation vxlan;
multicast-mode ingress-replication;

lab@DC2-SPINE2> show configuration protocols evpn | display set
set protocols evpn encapsulation vxlan
set protocols evpn multicast-mode ingress-replication
```

Switch Options

```
lab@DC2-SPINE2> show configuration switch-options
vtep-source-interface lo0.0;
route-distinguisher 10.0.255.6:1;
vrf-import EVPN_IMPORT;
vrf-target target:9999:9999;

lab@DC2-SPINE2> show configuration switch-options | display set
set switch-options vtep-source-interface lo0.0
set switch-options route-distinguisher 10.0.255.6:1
set switch-options vrf-import EVPN_IMPORT
set switch-options vrf-target target:9999:9999
```

EVPN Import Policy

```
lab@DC2-SPINE2> show configuration policy-options policy-statement EVPN_IMPORT
term import_ESI {
    from community ESI;
    then accept;
}
term last {
    then reject;
}

lab@DC2-SPINE2> show configuration policy-options policy-statement EVPN_IMPORT | display set
set policy-options policy-statement EVPN_IMPORT term import_ESI from community ESI
set policy-options policy-statement EVPN_IMPORT term import_ESI then accept
set policy-options policy-statement EVPN_IMPORT term last then reject
```

ESI Community

```
lab@DC2-SPINE2> show configuration policy-options community ESI
members target:9999:9999;
```

```
lab@DC2-SPINE2> show configuration policy-options community ESI | display set
set policy-options community ESI members target:9999:9999
```

DC2-LEAF1

BGP Overly

```
lab@DC2-LEAF1> show configuration protocols bgp group overlay-evpn
type internal;
local-address 10.0.255.7;
family evpn {
    signaling;
}
multipath;
neighbor 10.0.255.5;
neighbor 10.0.255.6;

lab@DC2-LEAF1> show configuration protocols bgp group overlay-evpn | display set
set protocols bgp group overlay-evpn type internal
set protocols bgp group overlay-evpn local-address 10.0.255.7
set protocols bgp group overlay-evpn family evpn signaling
set protocols bgp group overlay-evpn local-as 65002
set protocols bgp group overlay-evpn multipath
set protocols bgp group overlay-evpn neighbor 10.0.255.5
set protocols bgp group overlay-evpn neighbor 10.0.255.6
```

EVPN

```
lab@DC2-LEAF1> show configuration protocols evpn
encapsulation vxlan;
multicast-mode ingress-replication;

lab@DC2-LEAF1> show configuration protocols evpn | display set
set protocols evpn encapsulation vxlan
set protocols evpn multicast-mode ingress-replication
```

Switch Options

```
lab@DC2-LEAF1> show configuration switch-options
vtep-source-interface lo0.0;
route-distinguisher 10.0.255.7:1;
vrf-import EVPN_IMPORT;
vrf-target target:9999:9999;

lab@DC2-LEAF1> show configuration switch-options | display set
set switch-options vtep-source-interface lo0.0
set switch-options route-distinguisher 10.0.255.7:1
set switch-options vrf-import EVPN_IMPORT
set switch-options vrf-target target:9999:9999
```

EVPN Import Policy

```
lab@DC2-LEAF1> show configuration policy-options policy-statement EVPN_IMPORT
term import_ESI {
    from community ESI;
    then accept;
}
term last {
    then reject;
}

lab@DC2-LEAF1> show configuration policy-options policy-statement EVPN_IMPORT | display set
set policy-options policy-statement EVPN_IMPORT term import_ESI from community ESI
set policy-options policy-statement EVPN_IMPORT term import_ESI then accept
set policy-options policy-statement EVPN_IMPORT term last then reject
```

ESI Community

```
lab@DC2-LEAF1> show configuration policy-options community ESI
members target:9999:9999;

lab@DC2-LEAF1> show configuration policy-options community ESI | display set
set policy-options community ESI members target:9999:9999
```

DC2-LEAF2

BGP Overlay

```
lab@DC2-LEAF2> show configuration protocols bgp group overlay-evpn
type internal;
local-address 10.0.255.8;
family evpn {
    signaling;
}
local-as 65002;
multipath;
neighbor 10.0.255.5;
neighbor 10.0.255.6;

lab@DC2-LEAF2> show configuration protocols bgp group overlay-evpn | display set
set protocols bgp group overlay-evpn type internal
set protocols bgp group overlay-evpn local-address 10.0.255.8
set protocols bgp group overlay-evpn family evpn signaling
set protocols bgp group overlay-evpn local-as 65002
set protocols bgp group overlay-evpn multipath
set protocols bgp group overlay-evpn neighbor 10.0.255.5
set protocols bgp group overlay-evpn neighbor 10.0.255.6
```

EVPN

```
lab@DC2-LEAF2> show configuration protocols evpn
encapsulation vxlan;
multicast-mode ingress-replication;

lab@DC2-LEAF2> show configuration protocols evpn | display set
set protocols evpn encapsulation vxlan
set protocols evpn multicast-mode ingress-replication
```

Switch Options

```
lab@DC2-LEAF2> show configuration switch-options
vtep-source-interface lo0.0;
route-distinguisher 10.0.255.8:1;
vrf-import EVPN_IMPORT;
vrf-target target:9999:9999;

lab@DC2-LEAF2> show configuration switch-options | display set
set switch-options vtep-source-interface lo0.0
set switch-options route-distinguisher 10.0.255.8:1
set switch-options vrf-import EVPN_IMPORT
set switch-options vrf-target target:9999:9999
```

EVPN Import Policy

```
lab@DC2-LEAF2> show configuration policy-options policy-statement EVPN_IMPORT
term import_ESI {
    from community ESI;
    then accept;
}
term last {
    then reject;
}

lab@DC2-LEAF2> show configuration policy-options policy-statement EVPN_IMPORT | display set
set policy-options policy-statement EVPN_IMPORT term import_ESI from community ESI
set policy-options policy-statement EVPN_IMPORT term import_ESI then accept
set policy-options policy-statement EVPN_IMPORT term last then reject
```

ESI Community

```
lab@DC2-LEAF2> show configuration policy-options community ESI
members target:9999:9999;

lab@DC2-LEAF2> show configuration policy-options community ESI | display set
set policy-options community ESI members target:9999:9999
```

Data Center Interconnect Overlay Network

The only real difference between the data center overlay network and the data center interconnect overlay network is now we're using EBGP as opposed to IBGP with route reflection. The main reason for this is that each DC is assigned a unique ASN for administration purposes.

The other element to consider is that the data center interconnect overlay network is only used to exchange EVPN route type-5 between the spine switches.

NOTE Because we are not stretching Layer 2 between data centers, there is no problem with the default next hop behavior of EBGP. If Type-2 EVPN routes were required between data centers then we would need to use the `no-nexthop-change` knob.

Configure EBGP Overlay Network

Now let's create a new BGP group named *dci-overlay-evpn* and configure DC1-SPINE1 to peer with DC2-SPINE1, and likewise DC1-SPINE2 with DC2-SPINE2. Only family EVPN is required in the overlay and the peering is between loopback interfaces. Note that loopback information is exchanged using the underlay network.

DC1-SPINE1

```
lab@DC1-SPINE1> **show configuration protocols bgp group dci-overlay-evpn**
type external;
multihop;
local-address 10.0.255.1;
family evpn {
    signaling;
}
local-as 65103;
neighbor 10.0.255.5 {
    peer-as 65203;
}

lab@DC1-SPINE1> show configuration protocols bgp group dci-overlay-evpn | display set
set protocols bgp group dci-overlay-evpn type external
set protocols bgp group dci-overlay-evpn multihop
set protocols bgp group dci-overlay-evpn local-address 10.0.255.1
set protocols bgp group dci-overlay-evpn family evpn signaling
set protocols bgp group dci-overlay-evpn local-as 65103
set protocols bgp group dci-overlay-evpn neighbor 10.0.255.5 peer-as 65203
```

DC2-SPINE1

```
lab@DC2-SPINE1> show configuration protocols bgp group dci-overlay-evpn
type external;
multihop;
local-address 10.0.255.5;
family evpn {
    signaling;
}
local-as 65203;
neighbor 10.0.255.1 {
    peer-as 65103;
}

lab@DC2-SPINE1> show configuration protocols bgp group dci-overlay-evpn | display set
set protocols bgp group dci-overlay-evpn type external
set protocols bgp group dci-overlay-evpn multihop
set protocols bgp group dci-overlay-evpn local-address 10.0.255.5
set protocols bgp group dci-overlay-evpn family evpn signaling
set protocols bgp group dci-overlay-evpn local-as 65203
set protocols bgp group dci-overlay-evpn neighbor 10.0.255.1 peer-as 65103
```

DC1-SPINE2

```
lab@DC1-SPINE2> show configuration protocols bgp group dci-overlay-evpn
type external;
multihop;
local-address 10.0.255.2;
family evpn {
    signaling;
}
local-as 65104;
neighbor 10.0.255.6 {
    peer-as 65204;
}
```

```
lab@DC1-SPINE2> show configuration protocols bgp group dci-overlay-evpn | display set
set protocols bgp group dci-overlay-evpn type external
set protocols bgp group dci-overlay-evpn multihop
set protocols bgp group dci-overlay-evpn local-address 10.0.255.2
set protocols bgp group dci-overlay-evpn family evpn signaling
set protocols bgp group dci-overlay-evpn local-as 65104
set protocols bgp group dci-overlay-evpn neighbor 10.0.255.6 peer-as 65204
```

DC2-SPINE2

```
lab@DC2-SPINE2> show configuration protocols bgp group dci-overlay-evpn
type external;
multihop;
local-address 10.0.255.6;
family evpn {
    signaling;
}
local-as 65204;
neighbor 10.0.255.2 {
    peer-as 65104;
}

lab@DC2-SPINE2> show configuration protocols bgp group dci-overlay-evpn |display set
set protocols bgp group dci-overlay-evpn type external
set protocols bgp group dci-overlay-evpn multihop
set protocols bgp group dci-overlay-evpn local-address 10.0.255.6
set protocols bgp group dci-overlay-evpn family evpn signaling
set protocols bgp group dci-overlay-evpn local-as 65204
set protocols bgp group dci-overlay-evpn neighbor 10.0.255.2 peer-as 65104
```

Configure Tenant

Now it's time to configure a tenant as part of the overlay. In this recipe we are using Tenant3 host1, host2, host8, and host9. Each host resides in its own virtual network, but all are members of the same tenancy. Each virtual network is enabled with a VGA to provide gateway for the hosts. The IRB interfaces used for gateway are placed in a dedicated VRF for tenent3 in order to support multi-tenancy. For a detailed explanation of the various configuration steps to enable a tenant, please refer to Recipe 8.

Configure Tenant3 in DC1 and DC2

The various configuration elements required for enabling Tenant3 hosts T3-1, T3-2, T3-8, and T3-9 are included below. This does not include the EVPN type-5 steps, which will be subsequently covered.

DC1-LEAF1

Access Interface

```
lab@DC1-LEAF1> show configuration interfaces xe-0/0/3
description "t3-1 ens192";
unit 0 {
    family ethernet-switching {
        interface-mode access;
        vlan {
            members T3-1;
        }
    }
}

lab@DC1-LEAF1> show configuration interfaces xe-0/0/3 | display set
set interfaces xe-0/0/3 description "t3-1 ens192"
set interfaces xe-0/0/3 unit 0 family ethernet-switching interface-mode access
set interfaces xe-0/0/3 unit 0 family ethernet-switching vlan members T3-1
```

VXLAN

```
lab@DC1-LEAF1> show configuration vlans T3-1
vlan-id 301;
vxlan {
    vni 301;
    ingress-node-replication;
}

lab@DC1-LEAF1> show configuration vlans T3-1 | display set
set vlans T3-1 vlan-id 301
set vlans T3-1 vxlan vni 301
set vlans T3-1 vxlan ingress-node-replication
```

EVPN

```
lab@DC1-LEAF1> show configuration protocols evpn
encapsulation vxlan;
extended-vni-list [ 301 ];
multicast-mode ingress-replication;
vni-options {
    vni 301 {
        vrf-target export target:1:301;
    }
}

lab@DC1-LEAF1> show configuration protocols evpn | display set
set protocols evpn encapsulation vxlan
set protocols evpn extended-vni-list 301
set protocols evpn multicast-mode ingress-replication
set protocols evpn vni-options vni 301 vrf-target export target:1:301
```

EVPN Import Policy

```
lab@DC1-LEAF1> show configuration policy-options policy-statement EVPN_IMPORT
term import_T3-1 {
    from community T3-1;
    then accept;
}
term import_ESI {
    from community ESI;
    then accept;
}
term last {
    then reject;
}
```

```
lab@DC1-LEAF1> show configuration policy-options policy-statement EVPN_IMPORT | display set
set policy-options policy-statement EVPN_IMPORT term import_T3-1 from community T3-1
set policy-options policy-statement EVPN_IMPORT term import_T3-1 then accept
set policy-options policy-statement EVPN_IMPORT term import_ESI from community ESI
set policy-options policy-statement EVPN_IMPORT term import_ESI then accept
set policy-options policy-statement EVPN_IMPORT term last then reject
```

T3-1 Community

```
lab@DC1-LEAF1> show configuration policy-options community T3-1
members target:1:301;

lab@DC1-LEAF1> show configuration policy-options community T3-1 | display set
set policy-options community T3-1 members target:1:301
```

DC1-LEAF2

Access Interface

```
lab@DC1-LEAF2> show configuration interfaces xe-0/0/3
description "t3-8 ens192";
unit 0 {
    family ethernet-switching {
        interface-mode access;
        vlan {
            members T3-8;
        }
    }
}

lab@DC1-LEAF2> show configuration interfaces xe-0/0/3 | display set
set interfaces xe-0/0/3 description "t3-8 ens192"
set interfaces xe-0/0/3 unit 0 family ethernet-switching interface-mode access
set interfaces xe-0/0/3 unit 0 family ethernet-switching vlan members T3-8
```

VXLAN

```
lab@DC1-LEAF2> show configuration vlans T3-8
vlan-id 308;
vxlan {
    vni 308;
    ingress-node-replication;
}

lab@DC1-LEAF2> show configuration vlans T3-8 | display set
set vlans T3-8 vlan-id 308
set vlans T3-8 vxlan vni 308
set vlans T3-8 vxlan ingress-node-replication
```

EVPN

```
lab@DC1-LEAF2> show configuration protocols evpn
encapsulation vxlan;
extended-vni-list [308 ];
multicast-mode ingress-replication;
vni-options {
    vni 308 {
        vrf-target export target:1:308;
    }
}
```

```
lab@DC1-LEAF2> show configuration protocols evpn | display set
set protocols evpn encapsulation vxlan
set protocols evpn extended-vni-list 308
set protocols evpn multicast-mode ingress-replication
set protocols evpn vni-options vni 308 vrf-target export target:1:308
```

EVPN Import Policy

```
lab@DC1-LEAF2> show configuration policy-options policy-statement EVPN_IMPORT
term import_T3-8 {
    from community T3-8;
    then accept;
}
term import_ESI {
    from community ESI;
    then accept;
}
term last {
    then reject;
}

lab@DC1-LEAF2> show configuration policy-options policy-statement EVPN_IMPORT | display set
set policy-options policy-statement EVPN_IMPORT term import_T3-8 from community T3-8
set policy-options policy-statement EVPN_IMPORT term import_T3-8 then accept
set policy-options policy-statement EVPN_IMPORT term import_ESI from community ESI
set policy-options policy-statement EVPN_IMPORT term import_ESI then accept
set policy-options policy-statement EVPN_IMPORT term last then reject
```

T3-8 Community

```
lab@DC1-LEAF2> show configuration policy-options community T3-8
members target:1:308;

lab@DC1-LEAF2> show configuration policy-options community T3-8 | display set
set policy-options community T3-8 members target:1:308
```

DC1-SPINE1

VXLAN

```
lab@DC1-SPINE1> show configuration vlans
T3-1 {
    vlan-id 301;
    l3-interface irb.301;
    vxlan {
        vni 301;
        ingress-node-replication;
    }
}
T3-8 {
    vlan-id 308;
    l3-interface irb.308;
    vxlan {
        vni 308;
        ingress-node-replication;
    }
}
```

```
lab@DC1-SPINE1> show configuration vlans | display set
set vlans T3-1 vlan-id 301
set vlans T3-1 l3-interface irb.301
set vlans T3-1 vxlan vni 301
set vlans T3-1 vxlan ingress-node-replication
set vlans T3-8 vlan-id 308
set vlans T3-8 l3-interface irb.308
set vlans T3-8 vxlan vni 308
set vlans T3-8 vxlan ingress-node-replication
```

EVPN

```
lab@DC1-SPINE1> show configuration protocols evpn
encapsulation vxlan;
extended-vni-list [ 301 308 ];
multicast-mode ingress-replication;
default-gateway no-gateway-community;
vni-options {
    vni 301 {
        vrf-target export target:1:301;
    }
    vni 308 {
        vrf-target export target:1:308;
    }
}

lab@DC1-SPINE1> show configuration protocols evpn | display set
set protocols evpn encapsulation vxlan
set protocols evpn extended-vni-list 301
set protocols evpn extended-vni-list 308
set protocols evpn multicast-mode ingress-replication
set protocols evpn default-gateway no-gateway-community
set protocols evpn vni-options vni 301 vrf-target export target:1:301
set protocols evpn vni-options vni 308 vrf-target export target:1:308
```

EVPN Import Policy

```
lab@DC1-SPINE1> show configuration policy-options policy-statement EVPN_IMPORT
term import_T3-1 {
    from community T3-1;
    then accept;
}
term import_T3-8 {
    from community T3-8;
    then accept;
}
term import_ESI {
    from community ESI;
    then accept;
}
term last {
    then reject;
}

lab@DC1-SPINE1> show configuration policy-options policy-statement EVPN_IMPORT | display set
set policy-options policy-statement EVPN_IMPORT term import_T3-1 from community T3-1
set policy-options policy-statement EVPN_IMPORT term import_T3-1 then accept
set policy-options policy-statement EVPN_IMPORT term import_T3-8 from community T3-8
set policy-options policy-statement EVPN_IMPORT term import_T3-8 then accept
set policy-options politcy-statement EVPN_IMPORT term import_ESI from community ESI
set policy-options policy-statement EVPN_IMPORT term import_ESI then accept
set policy-options policy-statement EVPN_IMPORT term last then reject
```

T3-1 Community

```
lab@DC1-SPINE1> show configuration policy-options community T3-1
members target:1:301;

lab@DC1-SPINE1> show configuration policy-options community T3-1 | display set
set policy-options community T3-1 members target:1:301
```

T3-8 Community

```
lab@DC1-SPINE1> show configuration policy-options community T3-8
members target:1:308;

lab@DC1-SPINE1> show configuration policy-options community T3-8 | display set
set policy-options community T3-8 members target:1:308
```

IRB Interface

```
lab@DC1-SPINE1> show configuration interfaces irb
unit 301 {
    proxy-macip-advertisement;
    description " * T3 - vlan 301 - vni 301 ";
    family inet {
        address 192.168.31.252/24 {
            virtual-gateway-address 192.168.31.254;
        }
    }
}
unit 308 {
    proxy-macip-advertisement;
    description " * T3 - vlan 308 - vni 308 ";
    family inet {
        address 192.168.38.252/24 {
            virtual-gateway-address 192.168.38.254;
        }
    }
}

lab@DC1-SPINE1> show configuration interfaces irb | display set
set interfaces irb unit 301 proxy-macip-advertisement
set interfaces irb unit 301 description " * T3 - vlan 301 - vni 301 "
set interfaces irb unit 301 family inet address 192.168.31.252/24 virtual-gateway-
address 192.168.31.254
set interfaces irb unit 308 proxy-macip-advertisement
set interfaces irb unit 308 description " * T3 - vlan 308 - vni 308 "
set interfaces irb unit 308 family inet address 192.168.38.252/24 virtual-gateway-
address 192.168.38.254
```

Tenant 3 VRF

```
lab@DC1-SPINE1> show configuration routing-instances TENANT_3_VRF
instance-type vrf;
interface irb.301;
interface irb.308;
interface lo0.30;
route-distinguisher 10.0.255.1:30;
vrf-target target:1:300;

lab@DC1-SPINE1> show configuration routing-instances TENANT_3_VRF | display set
set routing-instances TENANT_3_VRF instance-type vrf
set routing-instances TENANT_3_VRF interface irb.301
set routing-instances TENANT_3_VRF interface irb.308
set routing-instances TENANT_3_VRF interface lo0.30
set routing-instances TENANT_3_VRF route-distinguisher 10.0.255.1:30
set routing-instances TENANT_3_VRF vrf-target target:1:300
```

DC1-SPINE2

VXLAN

```
lab@DC1-SPINE2> show configuration vlans
T3-1 {
    vlan-id 301;
    l3-interface irb.301;
    vxlan {
        vni 301;
        ingress-node-replication;
    }
}
T3-8 {
    vlan-id 308;
    l3-interface irb.308;
    vxlan {
        vni 308;
        ingress-node-replication;
    }
}

lab@DC1-SPINE2> show configuration vlans | display set
set vlans T3-1 vlan-id 301
set vlans T3-1 l3-interface irb.301
set vlans T3-1 vxlan vni 301
set vlans T3-1 vxlan ingress-node-replication
set vlans T3-8 vlan-id 308
set vlans T3-8 l3-interface irb.308
set vlans T3-8 vxlan vni 308
set vlans T3-8 vxlan ingress-node-replication
```

EVPN

```
lab@DC1-SPINE2> show configuration protocols evpn
encapsulation vxlan;
extended-vni-list [ 301 308 ];
multicast-mode ingress-replication;
default-gateway no-gateway-community;
vni-options {
    vni 301 {
        vrf-target export target:1:301;
    }
    vni 308 {
        vrf-target export target:1:308;
    }
}
```

```
lab@DC1-SPINE2> show configuration protocols evpn | display set
set protocols evpn encapsulation vxlan
set protocols evpn extended-vni-list 301
set protocols evpn extended-vni-list 308
set protocols evpn multicast-mode ingress-replication
set protocols evpn default-gateway no-gateway-community
set protocols evpn vni-options vni 301 vrf-target export target:1:301
set protocols evpn vni-options vni 308 vrf-target export target:1:308
```

EVPN Import Policy

```
lab@DC1-SPINE2> show configuration policy-options policy-statement EVPN_IMPORT
term import_T3-1 {
    from community T3-1;
    then accept;
}
term import_T3-8 {
    from community T3-8;
    then accept;
}
term import_ESI {
    from community ESI;
    then accept;
}
term last {
    then reject;
}

lab@DC1-SPINE2> show configuration policy-options policy-statement EVPN_IMPORT | display set
set policy-options policy-statement EVPN_IMPORT term import_T3-1 from community T3-1
set policy-options policy-statement EVPN_IMPORT term import_T3-1 then accept
set policy-options policy-statement EVPN_IMPORT term import_T3-8 from community T3-8
set policy-options policy-statement EVPN_IMPORT term import_T3-8 then accept
set policy-options policy-statement EVPN_IMPORT term import_ESI from community ESI
set policy-options policy-statement EVPN_IMPORT term import_ESI then accept
set policy-options policy-statement EVPN_IMPORT term last then reject
```

T3-1 Community

```
lab@DC1-SPINE2> show configuration policy-options community T3-1
members target:1:301;

lab@DC1-SPINE2> show configuration policy-options community T3-1 | display set
set policy-options community T3-1 members target:1:301
```

T3-8 Community

```
lab@DC1-SPINE2> show configuration policy-options community T3-8
members target:1:308;

lab@DC1-SPINE2> show configuration policy-options community T3-8 | display set
set policy-options community T3-8 members target:1:308
```

IRB Interface

```
lab@DC1-SPINE2> show configuration interfaces irb
unit 301 {
    proxy-macip-advertisement;
    description " * T3 - vlan 301 - vni 301 ";
    family inet {
        address 192.168.31.253/24 {
            virtual-gateway-address 192.168.31.254;
        }
    }
}
unit 308 {
    proxy-macip-advertisement;
    description " * T3 - vlan 308 - vni 308 ";
    family inet {
        address 192.168.38.253/24 {
            virtual-gateway-address 192.168.38.254;
        }
    }
}

lab@DC1-SPINE2> show configuration interfaces irb | display set
set interfaces irb unit 301 proxy-macip-advertisement
set interfaces irb unit 301 description " * T3 - vlan 301 - vni 301 "
set interfaces irb unit 301 family inet address 192.168.31.253/24 virtual-gateway-
address 192.168.31.254
set interfaces irb unit 308 proxy-macip-advertisement
set interfaces irb unit 308 description " * T3 - vlan 308 - vni 308 "
set interfaces irb unit 308 family inet address 192.168.38.253/24 virtual-gateway-
address 192.168.38.254
```

Tenant 3 VRF

```
lab@DC1-SPINE2> show configuration routing-instances TENANT_3_VRF
instance-type vrf;
interface irb.301;
interface irb.308;
interface lo0.30;
route-distinguisher 10.0.255.2:30;
vrf-target target:1:300;

lab@DC1-SPINE2> show configuration routing-instances TENANT_3_VRF | display set
set routing-instances TENANT_3_VRF instance-type vrf
set routing-instances TENANT_3_VRF interface irb.301
set routing-instances TENANT_3_VRF interface irb.308
set routing-instances TENANT_3_VRF interface lo0.30
set routing-instances TENANT_3_VRF route-distinguisher 10.0.255.2:30
set routing-instances TENANT_3_VRF vrf-target target:1:300
```

DC2-LEAF1

Access Interface

```
lab@DC2-LEAF1> show configuration interfaces xe-0/0/3
description "t3-2 ens192";
unit 0 {
    family ethernet-switching {
        interface-mode access;
        vlan {
            members T3-2;
        }
    }
}

lab@DC2-LEAF1> show configuration interfaces xe-0/0/3 | display set
set interfaces xe-0/0/3 description "t3-2 ens192"
set interfaces xe-0/0/3 unit 0 family ethernet-switching interface-mode access
set interfaces xe-0/0/3 unit 0 family ethernet-switching vlan members T3-2
```

VXLAN

```
lab@DC2-LEAF1> show configuration vlans T3-2
vlan-id 302;
vxlan {
    vni 302;
    ingress-node-replication;
}

lab@DC2-LEAF1> show configuration vlans T3-2 | display set
set vlans T3-2 vlan-id 302
set vlans T3-2 vxlan vni 302
set vlans T3-2 vxlan ingress-node-replication
```

EVPN

```
lab@DC2-LEAF1> show configuration protocols evpn
encapsulation vxlan;
extended-vni-list [ 302 ];
multicast-mode ingress-replication;
vni-options {
    vni 302 {
        vrf-target export target:1:302;
    }
}

lab@DC2-LEAF1> show configuration protocols evpn | display set
set protocols evpn encapsulation vxlan
set protocols evpn extended-vni-list 302
set protocols evpn multicast-mode ingress-replication
set protocols evpn vni-options vni 302 vrf-target export target:1:302
```

EVPN Import Policy

```
lab@DC2-LEAF1> show configuration policy-options policy-statement EVPN_IMPORT
term import_T3-2 {
    from community T3-2;
    then accept;
}
term import_ESI {
    from community ESI;
    then accept;
}
term last {
    then reject;
}
```

```
lab@DC2-LEAF1> show configuration policy-options policy-statement EVPN_IMPORT | display set
set policy-options policy-statement EVPN_IMPORT term import_T3-2 from community T3-2
set policy-options policy-statement EVPN_IMPORT term import_T3-2 then accept
set policy-options policy-statement EVPN_IMPORT term import_ESI from community ESI
set policy-options policy-statement EVPN_IMPORT term import_ESI then accept
set policy-options policy-statement EVPN_IMPORT term last then reject
```

T3-2 Community

```
lab@DC2-LEAF1> show configuration policy-options community T3-2
members target:1:302;

lab@DC2-LEAF1> show configuration policy-options community T3-2 | display set
set policy-options community T3-2 members target:1:302
```

DC2-LEAF2

Access Interface

```
lab@DC2-LEAF2> show configuration interfaces xe-0/0/3
description "t3-9 ens192";
unit 0 {
    family ethernet-switching {
        interface-mode access;
        vlan {
            members T3-9;
        }
    }
}

lab@DC2-LEAF2> show configuration interfaces xe-0/0/3 | display set
set interfaces xe-0/0/3 description "t3-9 ens192"
set interfaces xe-0/0/3 unit 0 family ethernet-switching interface-mode access
set interfaces xe-0/0/3 unit 0 family ethernet-switching vlan members T3-9
```

VXLAN

```
lab@DC2-LEAF2> show configuration vlans T3-9
vlan-id 309;
vxlan {
    vni 309;
    ingress-node-replication;
}

lab@DC2-LEAF2> show configuration vlans T3-9 | display set
set vlans T3-9 vlan-id 309
set vlans T3-9 vxlan vni 309
set vlans T3-9 vxlan ingress-node-replication
```

EVPN

```
lab@DC2-LEAF2> show configuration protocols evpn
encapsulation vxlan;
extended-vni-list [ 309 ];
multicast-mode ingress-replication;
vni-options {
    vni 309 {
        vrf-target export target:1:309;
    }
}
```

```
lab@DC2-LEAF2> show configuration protocols evpn | display set
set protocols evpn encapsulation vxlan
set protocols evpn extended-vni-list 309
set protocols evpn multicast-mode ingress-replication
set protocols evpn vni-options vni 309 vrf-target export target:1:309
```

EVPN Import Policy

```
lab@DC2-LEAF2> show configuration policy-options policy-statement EVPN_IMPORT
term import_T1-3 {
    from community T1-3;
    then accept;
}
term import_T1-4 {
    from community T1-4;
    then accept;
}
term import_T2-3 {
    from community T2-3;
    then accept;
}
term import_T2-4 {
    from community T2-4;
    then accept;
}
term import_T3-9 {
    from community T3-9;
    then accept;
}
term import_ESI {
    from community ESI;
    then accept;
}
term last {
    then reject;
}

lab@DC2-LEAF2> show configuration policy-options policy-statement EVPN_IMPORT | display set
set policy-options policy-statement EVPN_IMPORT term import_T1-3 from community T1-3
set policy-options policy-statement EVPN_IMPORT term import_T1-3 then accept
set policy-options policy-statement EVPN_IMPORT term import_T1-4 from community T1-4
set policy-options policy-statement EVPN_IMPORT term import_T1-4 then accept
set policy-options policy-statement EVPN_IMPORT term import_T2-3 from community T2-3
set policy-options policy-statement EVPN_IMPORT term import_T2-3 then accept
set policy-options policy-statement EVPN_IMPORT term import_T2-4 from community T2-4
set policy-options policy-statement EVPN_IMPORT term import_T2-4 then accept
set policy-options policy-statement EVPN_IMPORT term import_T3-9 from community T3-9
set policy-options policy-statement EVPN_IMPORT term import_T3-9 then accept
set policy-options policy-statement EVPN_IMPORT term import_ESI from community ESI
set policy-options policy-statement EVPN_IMPORT term import_ESI then accept
set policy-options policy-statement EVPN_IMPORT term last then reject
```

T3-9 Community

```
lab@DC2-LEAF2> show configuration policy-options community T3-9
members target:1:309;

lab@DC2-LEAF2> show configuration policy-options community T3-9 | display set
set policy-options community T3-9 members target:1:309
```

DC2-SPINE1

VXLAN

```
lab@DC2-SPINE1> show configuration vlans
T3-2 {
    vlan-id 302;
    l3-interface irb.302;
    vxlan {
        vni 302;
        ingress-node-replication;
    }
}
T3-9 {
    vlan-id 309;
    l3-interface irb.309;
    vxlan {
        vni 309;
        ingress-node-replication;
    }
}

lab@DC2-SPINE1> show configuration vlans | display set
set vlans T3-2 vlan-id 302
set vlans T3-2 l3-interface irb.302
set vlans T3-2 vxlan vni 302
set vlans T3-2 vxlan ingress-node-replication
set vlans T3-9 vlan-id 309
set vlans T3-9 l3-interface irb.309
set vlans T3-9 vxlan vni 309
set vlans T3-9 vxlan ingress-node-replication
```

EVPN

```
lab@DC2-SPINE1> show configuration protocols evpn
encapsulation vxlan;
extended-vni-list [ 103 104 203 204 302 309 ];
multicast-mode ingress-replication;
vni-options {
    vni 103 {
        vrf-target export target:1:103;
    }
    vni 104 {
        vrf-target export target:1:104;
    }
    vni 203 {
        vrf-target export target:1:203;
    }
    vni 204 {
        vrf-target export target:1:204;
    }
    vni 302 {
        vrf-target export target:1:302;
    }
    vni 309 {
        vrf-target export target:1:309;
    }
}
```

```
lab@DC2-SPINE1> **show configuration protocols evpn | display set**
set protocols evpn encapsulation vxlan
set protocols evpn extended-vni-list 302
set protocols evpn extended-vni-list 309
set protocols evpn multicast-mode ingress-replication
set protocols evpn vni-options vni 302 vrf-target export target:1:302
set protocols evpn vni-options vni 309 vrf-target export target:1:309
```

EVPN Import Policy

```
lab@DC2-SPINE1> show configuration policy-options policy-statement EVPN_IMPORT
term import_T3-2 {
    from community T3-2;
    then accept;
}
term import_T3-9 {
    from community T3-9;
    then accept;
}
term import_ESI {
    from community ESI;
    then accept;
}
term last {
    then reject;
}

lab@DC2-SPINE1> show configuration policy-options policy-statement EVPN_IMPORT | display set
set policy-options policy-statement EVPN_IMPORT term import_T3-2 from community T3-2
set policy-options policy-statement EVPN_IMPORT term import_T3-2 then accept
set policy-options policy-statement EVPN_IMPORT term import_T3-9 from community T3-9
set policy-options policy-statement EVPN_IMPORT term import_T3-9 then accept
set policy-options policy-statement EVPN_IMPORT term import_ESI from community ESI
set policy-options policy-statement EVPN_IMPORT term import_ESI then accept
set policy-options policy-statement EVPN_IMPORT term last then reject
```

T3-2 Community

```
lab@DC2-SPINE1> show configuration policy-options community T3-2
members target:1:302;

lab@DC2-SPINE1> show configuration policy-options community T3-2 | display set
set policy-options community T3-2 members target:1:302
```

T3-9 Community

```
lab@DC2-SPINE1> show configuration policy-options community T3-9
members target:1:309;

lab@DC2-SPINE1> show configuration policy-options community T3-9 | display set
set policy-options community T3-9 members target:1:309
```

IRB Interface

```
lab@DC2-SPINE1> show configuration interfaces irb
unit 302 {
    description " * T3 - vlan 302 - vni 302 ";
    proxy-arp;
    family inet {
        address 192.168.32.252/24 {
            virtual-gateway-address 192.168.32.254;
        }
    }
}
unit 309 {
    description " * T3 - vlan 309 - vni 309 ";
    proxy-arp;
    family inet {
        address 192.168.39.252/24 {
            virtual-gateway-address 192.168.39.254;
        }
    }
}

lab@DC2-SPINE1> show configuration interfaces irb | display set
set interfaces irb unit 302 description " * T3 - vlan 302 - vni 302 "
set interfaces irb unit 302 proxy-arp
set interfaces irb unit 302 family inet address 192.168.32.252/24 virtual-gateway-
address 192.168.32.254
set interfaces irb unit 309 description " * T3 - vlan 309 - vni 309 "
set interfaces irb unit 309 proxy-arp
set interfaces irb unit 309 family inet address 192.168.39.252/24 virtual-gateway-
address 192.168.39.254
```

Tenant 3 VRF

```
lab@DC2-SPINE1> show configuration routing-instances TENANT_3_VRF
instance-type vrf;
interface irb.302;
interface irb.309;
interface lo0.30;
route-distinguisher 10.0.255.5:30;
vrf-target target:1:300;

lab@DC2-SPINE1> show configuration routing-instances TENANT_3_VRF | display set
set routing-instances TENANT_3_VRF instance-type vrf
set routing-instances TENANT_3_VRF interface irb.302
set routing-instances TENANT_3_VRF interface irb.309
set routing-instances TENANT_3_VRF interface lo0.30
set routing-instances TENANT_3_VRF route-distinguisher 10.0.255.5:30
set routing-instances TENANT_3_VRF vrf-target target:1:300
```

DC2-SPINE2

VXLAN

```
lab@DC2-SPINE2> show configuration vlans
T3-2 {
    vlan-id 302;
    l3-interface irb.302;
    vxlan {
        vni 302;
        ingress-node-replication;
    }
}
```

```
T3-9 {
    vlan-id 309;
    l3-interface irb.309;
    vxlan {
        vni 309;
        ingress-node-replication;
    }
}

lab@DC2-SPINE2> show configuration vlans | display set
set vlans T3-2 vlan-id 302
set vlans T3-2 l3-interface irb.302
set vlans T3-2 vxlan vni 302
set vlans T3-2 vxlan ingress-node-replication
set vlans T3-9 vlan-id 309
set vlans T3-9 l3-interface irb.309
set vlans T3-9 vxlan vni 309
set vlans T3-9 vxlan ingress-node-replication
```

EVPN

```
lab@DC2-SPINE2> show configuration protocols evpn
encapsulation vxlan;
extended-vni-list [ 302 309 ];
multicast-mode ingress-replication;
vni-options {
    vni 302 {
        vrf-target export target:1:302;
    }
    vni 309 {
        vrf-target export target:1:309;
    }
}

lab@DC2-SPINE2> show configuration protocols evpn | display set
set protocols evpn encapsulation vxlan
set protocols evpn extended-vni-list 302
set protocols evpn extended-vni-list 309
set protocols evpn multicast-mode ingress-replication
set protocols evpn vni-options vni 302 vrf-target export target:1:302
set protocols evpn vni-options vni 309 vrf-target export target:1:309
```

EVPN Import Policy

```
lab@DC2-SPINE2> show configuration policy-options policy-statement EVPN_IMPORT
term import_T3-2 {
    from community T3-2;
    then accept;
}
term import_T3-9 {
    from community T3-9;
    then accept;
}
term import_ESI {
    from community ESI;
    then accept;
}
term last {
    then reject;
}
```

```
lab@DC2-SPINE2> show configuration policy-options policy-statement EVPN_IMPORT | display set
set policy-options policy-statement EVPN_IMPORT term import_T3-2 from community T3-2
set policy-options policy-statement EVPN_IMPORT term import_T3-2 then accept
set policy-options policy-statement EVPN_IMPORT term import_T3-9 from community T3-9
set policy-options policy-statement EVPN_IMPORT term import_T3-9 then accept
set policy-options policy-statement EVPN_IMPORT term import_ESI from community ESI
set policy-options policy-statement EVPN_IMPORT term import_ESI then accept
set policy-options policy-statement EVPN_IMPORT term last then reject
```

T3-2 Community

```
lab@DC2-SPINE2> show configuration policy-options community T3-2
members target:1:302;

lab@DC2-SPINE2> show configuration policy-options community T3-2 | display set
set policy-options community T3-2 members target:1:302
```

T3-9 Community

```
lab@DC2-SPINE2> show configuration policy-options community T3-9
members target:1:309;

lab@DC2-SPINE2> show configuration policy-options community T3-9 | display set
set policy-options community T3-9 members target:1:309
```

IRB Interface

```
lab@DC2-SPINE2> show configuration interfaces irb
unit 302 {
    description " * T3 - vlan 302 - vni 302 ";
    proxy-arp;
    family inet {
        address 192.168.32.253/24 {
            virtual-gateway-address 192.168.32.254;
        }
    }
}
unit 309 {
    description " * T3 - vlan 309 - vni 309 ";
    proxy-arp;
    family inet {
        address 192.168.39.253/24 {
            virtual-gateway-address 192.168.39.254;
        }
    }
}

lab@DC2-SPINE2> show configuration interfaces irb | display set
set interfaces irb unit 302 description " * T3 - vlan 302 - vni 302 "
set interfaces irb unit 302 proxy-arp
set interfaces irb unit 302 family inet address 192.168.32.253/24 virtual-gateway-
address 192.168.32.254
set interfaces irb unit 309 description " * T3 - vlan 309 - vni 309 "
set interfaces irb unit 309 proxy-arp
set interfaces irb unit 309 family inet address 192.168.39.253/24 virtual-gateway-
address 192.168.39.254
```

Tenant 3 VRF

```
lab@DC2-SPINE2> show configuration routing-instances TENANT_3_VRF
instance-type vrf;
interface irb.302;
interface irb.309;
interface lo0.30;
route-distinguisher 10.0.255.6:30;
vrf-target target:1:300;

lab@DC2-SPINE2> show configuration routing-instances TENANT_3_VRF | display set
set routing-instances TENANT_3_VRF instance-type vrf
set routing-instances TENANT_3_VRF interface irb.302
set routing-instances TENANT_3_VRF interface irb.309
set routing-instances TENANT_3_VRF interface lo0.30
set routing-instances TENANT_3_VRF route-distinguisher 10.0.255.6:30
set routing-instances TENANT_3_VRF vrf-target target:1:300
```

Configure EVPN Route Type-5

Now that Tenant3 is configured in both DC1 & DC2, and all hosts are able to reach their gateways, it's time to enable EVPN route type-5 for data center interconnect.

The IRB virtual gateways that were configured in the previous section are key. EVPN route type-5 requires a locally connected interface in order to generate the IP prefix. For example, the VRF routing table for Tenent3 must have an interface directly connected (IRB) for the IP prefix that we want to send via EVPN route type-5.

NOTE No changes or special configurations are required on the leaf switches for this solution.

Configure EVPN Route Type-5 on Spine Switches for Tenant-3

Configuring EVPN route type-5 is relatively simple and is completed under the `protocols evpn` hierarchy for a given tenant VRF.

DC1-SPINE1

```
lab@DC1-SPINE1> show configuration routing-instances TENANT_3_VRF
instance-type vrf;
interface irb.301;
interface irb.308;
interface lo0.30;
route-distinguisher 10.0.255.1:30;
vrf-target target:1:300;
protocols {
    evpn {
        ip-prefix-routes {
            advertise direct-nexthop;
            encapsulation vxlan;
            vni 5001;
        }
    }
}

lab@DC1-SPINE1> show configuration routing-instances TENANT_3_VRF | display set
set routing-instances TENANT_3_VRF instance-type vrf
set routing-instances TENANT_3_VRF interface irb.301
set routing-instances TENANT_3_VRF interface irb.308
set routing-instances TENANT_3_VRF interface lo0.30
set routing-instances TENANT_3_VRF route-distinguisher 10.0.255.1:30
set routing-instances TENANT_3_VRF vrf-target target:1:300
set routing-instances TENANT_3_VRF protocols evpn ip-prefix-routes advertise direct-nexthop
set routing-instances TENANT_3_VRF protocols evpn ip-prefix-routes encapsulation vxlan
set routing-instances TENANT_3_VRF protocols evpn ip-prefix-routes vni 5001
```

DC1-SPINE2

```
lab@DC1-SPINE2> show configuration routing-instances TENANT_3_VRF
instance-type vrf;
interface irb.301;
interface irb.308;
interface lo0.30;
route-distinguisher 10.0.255.2:30;
vrf-target target:1:300;
protocols {
    evpn {
        ip-prefix-routes {
            advertise direct-nexthop;
            encapsulation vxlan;
            vni 5001;
        }
    }
}

lab@DC1-SPINE2> show configuration routing-instances TENANT_3_VRF | display set
set routing-instances TENANT_3_VRF instance-type vrf
set routing-instances TENANT_3_VRF interface irb.301
set routing-instances TENANT_3_VRF interface irb.308
set routing-instances TENANT_3_VRF interface lo0.30
set routing-instances TENANT_3_VRF route-distinguisher 10.0.255.2:30
set routing-instances TENANT_3_VRF vrf-target target:1:300
set routing-instances TENANT_3_VRF protocols evpn ip-prefix-routes advertise direct-nexthop
set routing-instances TENANT_3_VRF protocols evpn ip-prefix-routes encapsulation vxlan
set routing-instances TENANT_3_VRF protocols evpn ip-prefix-routes vni 5001
```

DC2-SPINE1

```
lab@DC2-SPINE1> show configuration routing-instances TENANT_3_VRF
instance-type vrf;
interface irb.302;
interface irb.309;
interface lo0.30;
route-distinguisher 10.0.255.5:30;
vrf-target target:1:300;
protocols {
    evpn {
        ip-prefix-routes {
            advertise direct-nexthop;
            encapsulation vxlan;
            vni 5001;
        }
    }
}

lab@DC2-SPINE1> show configuration routing-instances TENANT_3_VRF | display set
set routing-instances TENANT_3_VRF instance-type vrf
set routing-instances TENANT_3_VRF interface irb.302
set routing-instances TENANT_3_VRF interface irb.309
set routing-instances TENANT_3_VRF interface lo0.30
set routing-instances TENANT_3_VRF route-distinguisher 10.0.255.5:30
set routing-instances TENANT_3_VRF vrf-target target:1:300
set routing-instances TENANT_3_VRF protocols evpn ip-prefix-routes advertise direct-nexthop
set routing-instances TENANT_3_VRF protocols evpn ip-prefix-routes encapsulation vxlan
set routing-instances TENANT_3_VRF protocols evpn ip-prefix-routes vni 5001
```

DC2-SPINE2

```
lab@DC2-SPINE2> show configuration routing-instances TENANT_3_VRF
instance-type vrf;
interface irb.302;
interface irb.309;
interface lo0.30;
route-distinguisher 10.0.255.6:30;
vrf-target target:1:300;
protocols {
    evpn {
        ip-prefix-routes {
            advertise direct-nexthop;
            encapsulation vxlan;
            vni 5001;
        }
    }
}

lab@DC2-SPINE2> show configuration routing-instances TENANT_3_VRF | display set
set routing-instances TENANT_3_VRF instance-type vrf
set routing-instances TENANT_3_VRF interface irb.302
set routing-instances TENANT_3_VRF interface irb.309
set routing-instances TENANT_3_VRF interface lo0.30
set routing-instances TENANT_3_VRF route-distinguisher 10.0.255.6:30
set routing-instances TENANT_3_VRF vrf-target target:1:300
set routing-instances TENANT_3_VRF protocols evpn ip-prefix-routes advertise direct-nexthop
set routing-instances TENANT_3_VRF protocols evpn ip-prefix-routes encapsulation vxlan
set routing-instances TENANT_3_VRF protocols evpn ip-prefix-routes vni 5001
```

Verification

In order to make the solution redundant, it is necessary to ensure that each spine switch retains reachability to the remote subnets, received via EVPN route type-5, in the event of data center interconnect failure. This is achieved by announcing the remote EVPN type-5 routes, received from the remote spine switch, to the peer spine switch within a given DC, as shown in Figure 9.4.

Under normal operation, DC1-SPINE2 receives a type-5 EVPN route for remote T3-2 and T3-9 subnets via DC2-SPINE2 and also from DC1-SPINE1. Traffic destined to the remote DC2 subnets traverses the locally connected DCI circuit due to a shorter AS-Path.

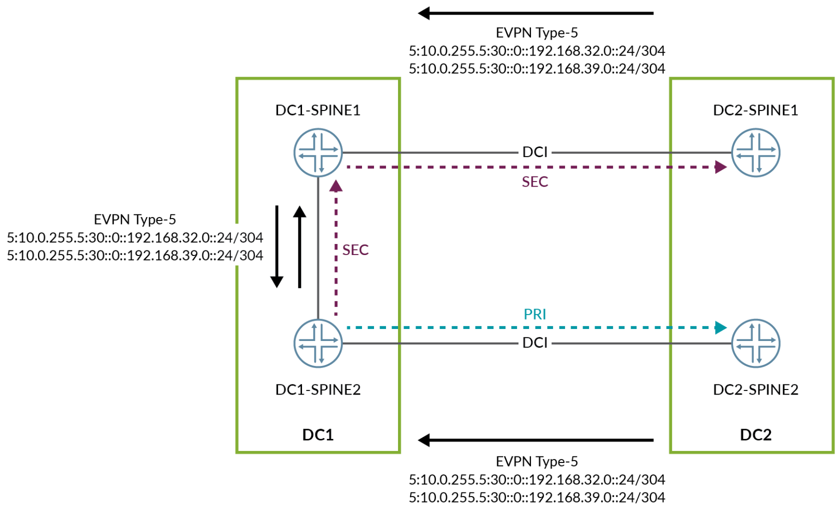


Figure 9.4 DCI Type-5 Routing No Failure

DC1-SPINE2 TENANT_3 Routing Table

```
lab@DC1-SPINE2> **show route table TENANT_3_VRF.evpn.0**

TENANT_3_VRF.evpn.0: 8 destinations, 12 routes (8 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

5:10.0.255.1:30::0::192.168.31.0::24/304
                   *[BGP/170] 1w0d 01:46:01, localpref 100, from 10.0.255.1
                      AS path: I, validation-state: unverified
                    > to 172.16.0.40 via xe-0/0/5.0
5:10.0.255.1:30::0::192.168.38.0::24/304
                   *[BGP/170] 1w0d 01:46:01, localpref 100, from 10.0.255.1
                      AS path: I, validation-state: unverified
                    > to 172.16.0.40 via xe-0/0/5.0
5:10.0.255.2:30::0::192.168.31.0::24/304
                   *[EVPN/170] 1w1d 05:38:42
                      Indirect
5:10.0.255.2:30::0::192.168.38.0::24/304
                   *[EVPN/170] 1w1d 05:38:42
                      Indirect
**5:10.0.255.5:30::0::192.168.32.0::24/304**
                   ***[BGP/170] 1d 01:29:19, localpref 100, from 10.0.255.6**
```

```
                      AS path: 65204 65002 I, validation-state: unverified
                    > to 172.16.99.3 via xe-0/0/4.0
                    [BGP/170] 08:50:11, localpref 100, from 10.0.255.1
                      AS path: 65103 65203 I, validation-state: unverified
                    > to 172.16.0.40 via xe-0/0/5.0
5:10.0.255.5:30::0::192.168.39.0::24/304
                   *[BGP/170] 1d 01:29:19, localpref 100, from 10.0.255.6
                      AS path: 65204 65002 I, validation-state: unverified
                    > to 172.16.99.3 via xe-0/0/4.0
                    [BGP/170] 08:50:11, localpref 100, from 10.0.255.1
                      AS path: 65103 65203 I, validation-state: unverified
                    > to 172.16.0.40 via xe-0/0/5.0
5:10.0.255.6:30::0::192.168.32.0::24/304
                   *[BGP/170] 1d 01:29:19, localpref 100, from 10.0.255.6
                      AS path: 65204 I, validation-state: unverified
                    > to 172.16.99.3 via xe-0/0/4.0
                    [BGP/170] 08:50:11, localpref 100, from 10.0.255.1
                      AS path: 65103 65203 65002 I, validation-state: unverified
                    > to 172.16.0.40 via xe-0/0/5.0
5:10.0.255.6:30::0::192.168.39.0::24/304
                   *[BGP/170] 1d 01:29:19, localpref 100, from 10.0.255.6
                      AS path: 65204 I, validation-state: unverified
                    > to 172.16.99.3 via xe-0/0/4.0
                    [BGP/170] 08:50:11, localpref 100, from 10.0.255.1
                      AS path: 65103 65203 65002 I, validation-state: unverified
                    > to 172.16.0.40 via xe-0/0/5.0
```

DC1-SPINE2 DC2-SPINE2 DCI Link Failure

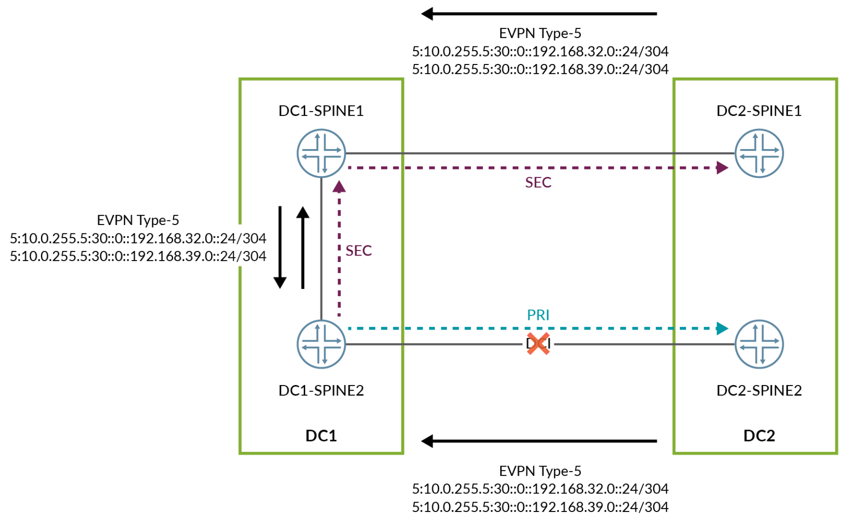


Figure 9.5 DCI Type-5 Routing Link Failure

In the event of a DCI failure between DC1-SPINE2 and DC2-SPINE2, shown in Figure 9.5, the route via DC2-SPINE2 is withdrawn and traffic is routed via DC1-SPINE1.

```
lab@DC1-SPINE2> show route table TENANT_3_VRF.evpn.0

TENANT_3_VRF.evpn.0: 8 destinations, 12 routes (8 active, 0 holddown, 4 hidden)
+ = Active Route, - = Last Active, * = Both

5:10.0.255.1:30::0::192.168.31.0::24/304
                   *[BGP/170] 1w0d 01:51:39, localpref 100, from 10.0.255.1
                      AS path: I, validation-state: unverified
                    > to 172.16.0.40 via xe-0/0/5.0
5:10.0.255.1:30::0::192.168.38.0::24/304
                   *[BGP/170] 1w0d 01:51:39, localpref 100, from 10.0.255.1
                      AS path: I, validation-state: unverified
                    > to 172.16.0.40 via xe-0/0/5.0
5:10.0.255.2:30::0::192.168.31.0::24/304
                   *[EVPN/170] 1w1d 05:44:20
                      Indirect
5:10.0.255.2:30::0::192.168.38.0::24/304
                   *[EVPN/170] 1w1d 05:44:20
                      Indirect
5:10.0.255.5:30::0::192.168.32.0::24/304
                   *[BGP/170] 00:00:54, localpref 100, from 10.0.255.1
                      AS path: 65103 65203 I, validation-state: unverified
                    > to 172.16.0.40 via xe-0/0/5.0
5:10.0.255.5:30::0::192.168.39.0::24/304
                   *[BGP/170] 00:00:54, localpref 100, from 10.0.255.1
                      AS path: 65103 65203 I, validation-state: unverified
                    > to 172.16.0.40 via xe-0/0/5.0
5:10.0.255.6:30::0::192.168.32.0::24/304
                   *[BGP/170] 00:00:54, localpref 100, from 10.0.255.1
                      AS path: 65103 65203 65002 I, validation-state: unverified
                    > to 172.16.0.40 via xe-0/0/5.0
5:10.0.255.6:30::0::192.168.39.0::24/304
                   *[BGP/170] 00:00:54, localpref 100, from 10.0.255.1
                      AS path: 65103 65203 65002 I, validation-state: unverified
                    > to 172.16.0.40 via xe-0/0/5.0
```

You can see in this example that the link between DC1-SPINE2 and DC2-SPINE2 has failed. However DC1-SPINE2 still has reachability to remote DC2 subnets via the EVPN type-5 routes received from DC1-SPINE1. This is important, particularly when using EVPN anycast gateway, whereby hosts may send traffic to either spine switch.

Discussion

In this recipe EVPN route type-5 was used to aggregate tenant EVPN type-2 MACs behind a single EVPN type-5 IP prefix for a given virtual network. The type-5 routes were then advertised over DCI interconnects to provide a redundant DCI setup.