



Cisco APIC REST API Configuration Guide

First Published: 2016-10-31

Last Modified: 2017-02-01

Americas Headquarters

Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
<http://www.cisco.com>
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 527-0883

© 2016-2017 Cisco Systems, Inc. All rights reserved.



CONTENTS

P r e f a c e

Preface **xix**

Audience **xix**

New and Changed Information **xix**

Document Conventions **xxi**

Related Documentation **xxiii**

Documentation Feedback **xxiii**

P A R T I

Part 1: Cisco APIC REST API Usage Guidelines **1**

C H A P T E R 1

Using the REST API **3**

About the REST API **3**

 About the REST API **3**

 Management Information Model **4**

 Object Naming **5**

 Composing REST API Requests **6**

 Read and Write Operations and Filters **6**

 Using Classes in REST API Commands **8**

 Using Managed Objects in REST API Commands **8**

 Creating the API Command **9**

 Composing the API Command Body **11**

 Composing the API Command Body to Call a Method **11**

 Composing the API Command Body for an API Operation on an MO **12**

 Using Tags and Alias **13**

 Composing REST API Queries **15**

 Composing Query Filter Expressions **15**

 Applying Query Scoping Filters **17**

 Filtering API Query Results **19**

Filter Conditional Operators	20
Sorting and Paginating Query Results	20
Subscribing to Query Results	21
REST API Examples	23
Information About the API Examples	23
Example: Using the JSON API to Add a Leaf Port Selector Profile	24
Example: Using the JSON API to Get Information About a Node	26
Example: Using the JSON API to Get Running Firmware	27
Example: Using the JSON API to Get Top Level System Elements	28
Example: Using the XML API and OwnerTag to Add Audit Log Information to Actions	29
Example: XML Get Endpoints (Devices) with IP and MAC Addresses	29
Example: Monitoring Using the REST API	30
Accessing the REST API	30
Accessing the REST API	30
Invoking the API	30
Configuring the HTTP Request Method and Content Type	31
Configuring HTTP and HTTPS Using the GUI	31
Configuring a Custom Certificate for Cisco ACI HTTPS Access Using the GUI	32
Authenticating and Maintaining an API Session	34
Requiring a Challenge Token for an API Session	35
Logging In	36
Changing Your Own User Credentials	36
REST API Tools	38
Management Information Model Reference	38
Viewing an API Interchange in the GUI	39
Testing the API Using Browser Add-Ons	41
Testing the API with cURL	42
Cisco APIC Python SDK	42
Using the Managed Object Browser (Visore)	43
Visore Browser Page	43
Accessing Visore	44
Running a Query in Visore	45

CHAPTER 2**Managing APIC Using the REST API 49****Adding Management Access 49****In-Band and Out-of-Band Management Access 49****About Static Management Access 49****Configuring In-Band Management Access Using the REST API 50****Configuring Static In-Band Management Access Using the REST API 53****Configuring Out-of-Band Management Access Using the REST API 55****Configuring Static Out-of-Band Management Access Using the REST API 57****Managing Configuration Files 58****Overview 58****Backing Up, Restoring, and Rolling Back Configuration Files Workflow 59****About Configuration Export to Controllers 59****About Configuration Import to Controller 60****Configuration File Encryption 61****About the fileRemotePath Object 62****Configuring a Remote Location Using the REST API 63****Configuring Configuration File Export to Controller Using the REST API 63****Configuring a Configuration File Import Policy Using the REST API 63****Encrypting Configuration Files Using the REST API 64****Snapshots and Rollbacks 64****Snapshots 64****About Rollbacks 64****Uploading and Downloading Snapshots Using the REST API 65****Configuring and Executing a Configuration Rollback Using the REST API 66****Using Configuration Zones 66****Configuration Zones 66****Configuration Zone Supported Policies 67****Creating Configuration Zones Using the REST API 69**

CHAPTER 3**Managing Roles, Users, and Signature-Based Transactions 71****Managing APIC Roles and Users 71****User Access, Authorization, and Accounting 71****Access Rights Workflow Dependencies 71****Accounting 71**

Multiple Tenant Support	72
User Access: Roles, Privileges, and Security Domains	73
Configuring a Custom Role Using the REST API	74
Configuring a Local User	74
Configuring a Local User Using the REST API	75
Configuring a Remote User	75
Configuring a Remote User Using the REST API	75
APIC Signature-Based Transactions	76
About Signature-Based Transactions	76
Using a Private Key to Calculate a Signature	76
Guidelines and Limitations	78
Creating a Local User and Adding a User Certificate Using the REST API	78

CHAPTER 4

Common Tenant Tasks 81

Common Tenant Tasks	81
Tenants Overview	81
Tenant Creation	81
Adding a Tenant	81
Example: Using the JSON API to Add a Tenant	82
Example: Using the XML API to Add a Tenant	83

CHAPTER 5

Managing Layer 2 Networking 85

Tenant External Bridged Networks	85
Bridged Interface to an External Router	85
VRF and Bridge Domains	86
Creating a Tenant, VRF, and Bridge Domain Using the REST API	86
Ports	86
Statically Deploying an EPG on a Specific Port	86
Deploying an EPG on a Specific Port with APIC Using the REST API	87
Creating Domains, Attach Entity Profiles, and VLANs to Deploy an EPG on a Specific Port	87
Creating AEP, Domains, and VLANs to Deploy an EPG on a Specific Port Using the REST API	87
Creating a Port Channel Policy Using the REST API	89

CHAPTER 6**Managing Layer 3 Networking 91**

- Configuring External Connectivity for Tenants 91
- Configuring a Tenant Layer 3 Outside Network Connection Overview 91
- Configuring Layer 3 Outside for Tenant Networks Using the REST API 91

CHAPTER 7**Monitoring Using the REST API 93**

- About Monitoring Using the REST API 93
- Monitoring APIC Using the REST API 93
 - APIC 93
 - Monitoring APIC CPU and Memory Usage Using the REST API 93
 - Monitoring APIC Disk Utilization Using the REST API 94
 - Monitoring Physical Interface Statistics and Link State Using the REST API 94
 - Fabric 95
 - Monitoring LLDP and CDP Neighbor Status Using the REST API 95
 - Monitoring Physical and Bond Interfaces Using the REST API 95
 - Monitoring EPG-Level Statistics Using the REST API 95
 - Switches 96
 - Monitoring Switch CPU Utilization Using the REST API 96
 - Monitoring Switch Fan Status Using the REST API 97
 - Monitoring Switch Memory Utilization Using the REST API 97
 - Monitoring Switch Module Status Using the REST API 98
 - Monitoring Switch Power Supply Status Using the REST API 98
 - Monitoring Switch Inventory Using the REST API 98

CHAPTER 8**Troubleshooting Using the REST API 99**

- Collecting and Exporting Technical Support Information 99
 - About Exporting Files 99
 - Sending an On-Demand TechSupport File Using the REST API 100
- Troubleshooting Using Atomic Counters 100
 - Atomic Counters 100
 - Enabling Atomic Counters 101
 - Troubleshooting Using Atomic Counters with the REST API 102
- Troubleshooting Using Faults 103
 - Understanding APIC Faults 103

Troubleshooting Using Faults with the REST API **103**

Statistics **105**

Configuring a Stats Monitoring Policy Using the REST API **105**

Recovering a Disconnected Leaf **106**

Recovering a Disconnected Leaf **106**

Recovering a Disconnected Leaf Using the REST API **106**

Troubleshooting Contracts and Taboo Contracts with Permit and Deny Logging **107**

Verifying Contracts, Taboo Contracts, and Filters Using the REST API **107**

Viewing ACL Permit and Deny Logs Using the REST API **107**

Troubleshooting Using Digital Optical Monitoring Statistics **108**

Troubleshooting Using Digital Optical Monitoring With the REST API **108**

Troubleshooting Using Port Tracking **109**

Port Tracking Policy for Uplink Failure Detection **109**

Port Tracking Using the REST API **109**

Removing Unwanted _ui_ Objects **110**

Removing Unwanted _ui_ Objects Using the REST API **110**

PART III

Part 3: Setting Up APIC and the Fabric Using the REST API **111**

CHAPTER 9

Managing APIC Clusters **113**

Cluster Management Guidelines **113**

Cluster Management Guidelines **113**

Expanding and Contracting Clusters **114**

Expanding the APIC Cluster Size **114**

Expanding the Cisco APIC Cluster **114**

Expanding the APIC Cluster Using the REST API **115**

Contracting the Cisco APIC Cluster **115**

Contracting the APIC Cluster Using the REST API **115**

Managing Cluster High Availability **116**

About High Availability for APIC Cluster **116**

Switching Over Active APIC with Standby APIC Using REST API **117**

CHAPTER 10

Configuring Tenant Policies **119**

Basic Tenant Configuration **119**

Creating a Tenant, VRF, and Bridge Domain Using the REST API **119**

Tenants in Multiple Private Networks	120
About Multiple Private Networks with Inter-Tenant Communication	120
Configuring Multiple Private Networks with Inter-Tenant Communication Using the REST API	120
About Multiple Private Networks with Intra-Tenant Communication	122
Configuring Multiple Tenants with Intra-Tenant Communication Using the REST API	123
Tenant Policy Example	123
Tenant Policy Example Overview	123
Tenant Policy Example XML Code	124
Tenant Policy Example Explanation	125
Policy Universe	125
Tenant Policy Example	126
Filters	126
Contracts	127
Subjects	128
Labels	128
VRF	129
Bridge Domains	129
Application Profiles	130
Endpoints and Endpoint Groups (EPGs)	130
Closing	132
What the Example Tenant Policy Does	132
EPGs	133
Deploying an Application EPG through an AEP or Interface Policy Group to Multiple Ports	133
Deploying an EPG on a Specific Port with APIC Using the REST API	133
Deploying an EPG through an AEP to Multiple Interfaces Using the REST API	134
Creating AEP, Domains, and VLANs to Deploy an EPG on a Specific Port Using the REST API	135
Intra-EPG Isolation	136
Intra-EPG Isolation for Bare Metal Servers	136
Configuring Intra-EPG Isolation for Bare Metal Servers Using the REST API	137
Intra-EPG Isolation for VMware vDS	138
Configuring Intra-EPG Isolation for VMware vDS using the REST API	140
Intra-EPG Isolation Enforcement for Cisco AVS	140

Configuring Intra-EPG Isolation for Cisco AVS Using the REST API	141
Microsegmentation	141
Using Microsegmentation with Network-based Attributes on Bare Metal	141
Configuring Microsegmentation with Cisco ACI Using the REST API	142
Configuring an IP-based Microsegmented EPG as a Shared Resource Using the REST API	143
Configuring a Network-Based Microsegmented EPG in a Bare-Metal Environment Using the REST API	143
Application Profiles	144
Three-Tier Application Deployment	144
Parameters to Create a Filter for http	145
Parameters to Create Filters for rmi and sql	146
Deploying an Application Profile Using the REST API	146
Contracts, Taboo Contracts, and Preferred Groups	148
Security Policy Enforcement	148
Contracts and Taboo Contracts	149
Contracts Contain Security Policy Specifications	149
Contracts	152
Configuring a Contract Using the REST API	153
Configuring a Taboo Contract Using the REST API	153
Contract Preferred Groups	154
About Contract Preferred Groups	154
Configuring Contract Preferred Groups Using the REST API	155

CHAPTER 11

Provisioning Core Services	157
DHCP	157
Configuring a DHCP Relay Policy	157
Configuring a DHCP Server Policy for the APIC Infrastructure Using the REST API	158
Layer 2 and Layer 3 DHCP Relay Sample Policies	158
DNS	160
DNS	160
Configuring a DNS Service Policy to Connect with DNS Providers Using the REST API	161
DNS Policy Example	161

NTP	162
Time Synchronization and NTP	162
Configuring NTP Using the REST API	162
Tetration	163
Overview	163
Configuring Cisco Tetration Analytics Using the REST API	163
NetFlow	164
About NetFlow	164
Configuring a NetFlow Exporter Policy for VM Networking Using the REST API	164
Configuring NetFlow Infra Selectors Using the REST API	164
Configuring the NetFlow Tenant Hierarchy Using the REST API	165
Consuming a NetFlow Exporter Policy Under a VMM Domain Using the REST API	167
Configuring the NetFlow or Tetration Analytics Priority Using the REST API	167
ACL Contract Permit and Deny Logging	167
About ACL Contract Permit and Deny Logs	167
Enabling Taboo Contract Deny Logging Using the REST API	168
DOM Statistics	168
About Digital Optical Monitoring	168
Enabling Digital Optical Monitoring Using the REST API	168
Syslog	170
About Syslog	170
Configuring a Syslog Group and Destination Using the REST API	171
Creating a Syslog Source Using the REST API	171
Enabling Syslog to Display in NX-OS CLI Format, Using the REST API	171
Data Plane Policing	172
Overview	172
Configuring Data Plane Policing Using the REST API	173
Traffic Storm Control	175
About Traffic Storm Control	175
Configuring a Traffic Storm Control Policy Using the REST API	175

CHAPTER 12**Provisioning Layer 2 Networks**

Networking Domains, VLANs, and AEPs

177

Networking Domains

177

Configuring a Physical Domain Using the REST API

Creating VLAN Pools	178
Creating a VLAN Pool Using the REST API	179
Attachable Entity Profile	179
Creating an Attachable Access Entity Profile Using the REST API	180
Interfaces	181
Ports, PCs, and VPCs	181
Configuring a Single Port Channel Applied to Multiple Switches	181
Configuring a Single Virtual Port Channel Across Two Switches Using the REST API	182
Configuring Two Port Channels Applied to Multiple Switches Using the REST API	183
Configuring a Virtual Port Channel on Selected Port Blocks of Two Switches Using the REST API	184
Configuring a Virtual Port Channel and Applying it to a Static Port Using the REST API	186
Interface Speed	187
Interface Configuration Guidelines	187
Changing Interface Speed	188
FEXs	189
ACI FEX Guidelines	189
Configuring an FEX VPC Policy Using the REST API	189
FCoE	191
Supporting Fibre Channel over Ethernet Traffic on the ACI Fabric	191
Configuring FCoE Connectivity Using the REST API	194
Configuring FCoE Over FEX Using REST API	197
Undeploying FCoE Connectivity through the REST API or SDK	201
802.1Q Tunnels	205
About ACI 802.1Q Tunnels	205
Configuring 802.1Q Tunnels With Ports Using the REST API	206
Configuring 802.1Q Tunnels With PCs Using the REST API	207
Configuring 802.1 Q Tunnels With VPCs Using the REST API	209
Breakout Ports	210
Configuration of Dynamic Breakout Ports	210
Configuring Dynamic Breakout Ports Using the REST API	211
IGMP Snooping	214

About Cisco APIC and IGMP Snooping	214
How IGMP Snooping is Implemented in the ACI Fabric	214
Virtualization Support	215
Configuring and Assigning an IGMP Snoop Policy to a Bridge Domain using the REST API	216
Enabling Group Access to IGMP Layer 2 Multicast using REST API	216
Enabling IGMP Layer 2 Multicast on Static Ports Using the REST API	217
Proxy ARP	218
About Proxy ARP	218
Guidelines and Limitations	226
Configuring Proxy ARP Using the REST API	227

CHAPTER 13

Provisioning Layer 3 Outside Connections	229
Layer 3 Outside Connections	229
Configuring a Tenant Layer 3 Outside Network Connection Overview	229
Configuring Layer 3 Outside for Tenant Networks Using the REST API	229
Tenant External Network Policy Example	230
Route Controls	232
Configuring a Routing Control Protocol Using Import and Export Controls	232
Configuring a Route Control Protocol to Use Import and Export Controls, With the REST API	232
BGP EVPN Services for Fabric WAN	234
Layer 3 EVPN Services Over Fabric WAN	234
Configuring Layer 3 EVPN for WAN Services Using the REST API	236
Distributing BGP EVPN Type-2 Host Routes to a DCIG	241
Enabling Distributing BGP EVPN Type-2 Host Routes to a DCIG Using the REST API	241
Multipod	242
Multipod	242
Setting Up Multipod Fabric Using the REST API	244
HSRP	247
About HSRP	247
Guidelines and Limitations	248
Configuring HSRP in APIC Using REST API	248
IP Multicast	250

Layer 3 Multicast	250
Guidelines for Configuring Layer 3 Multicast	251
Configuring Layer 3 Multicast Using REST API	252
Pervasive Gateway	254
Common Pervasive Gateway	254
Configuring Common Pervasive Gateway Using the REST API	255
Explicit Prefix Lists	255
About Explicit Prefix List Support for Route Maps/Profile	255
Guidelines and Limitations	257
About Route Map/Profile	258
Aggregation Support for Explicit Prefix List	258
Configuring Route Map/Profile with Explicit Prefix List Using REST API	259
IP Address Aging Tracking	260
Overview	260
Configuring IP Aging Using the REST API	260
Route Summarization	260
Configuring Route Summarization for BGP, OSPF, and EIGRP Using the REST API	260
Configuring Route Summarization for BGP, OSPF, and EIGRP Using the REST API	262
External Route Interleak	264
Overview	264
Configuring Interleak of External Routes Using the REST API	264
Routing Protocols	265
BGP and BFD	265
Guidelines for Configuring a BGP Layer 3 Outside Network Connection	265
BGP Connection Types and Loopback Guidelines	266
Configuring an MP-BGP Route Reflector Using the REST API	267
Configuring BGP External Routed Network Using the REST API	267
Configuring BFD Consumer Protocols Using the REST API	269
Configuring BFD Globally Using the REST API	270
Configuring BFD Interface Override Using the REST API	270
OSPF	271
OSPF Layer 3 Outside Connections	271

Creating OSPF External Routed Network for Management Tenant Using REST API **272**

EIGRP 273

Overview **273**

Configuring EIGRP Using the REST API **274**

Neighbor Discovery 276

Neighbor Discovery **276**

Creating the Tenant, VRF, and Bridge Domain with IPv6 Neighbor Discovery Using the REST API **277**

CHAPTER 14

Managing Layer 4 to Layer 7 Services 279

About Layer 4 to Layer 7 Services **279**

About Application-Centric Infrastructure Layer 4 to Layer 7 Services **279**

Access for Managing Layer 4 to Layer 7 Services **280**

Configure In-Band Connectivity to Devices Using Tenant's VRF Using the REST API **280**

Configuring In-Band Connectivity to Devices Using Management Tenant VRF Using the REST API **281**

Device Packages **283**

About the Device Package **283**

Notes for Installing a Device Package with REST **283**

Uploading a Device Package File Using the API **283**

Installing a Device Package Using the REST API **284**

Using an Imported Device with the REST APIs **284**

Trunking **284**

About Trunking **284**

Enabling Trunking on a Layer 4 to Layer 7 Virtual ASA device Using the REST APIs **285**

Device Selection Policies **285**

About Device Selection Policies **285**

Creating a Device Selection Policy Using the REST API **285**

Adding a Logical Interface in a Device Using the REST APIs **286**

Service Graph Templates **286**

About Service Graph Templates **286**

Configuring a Service Graph Template Using the REST APIs **287**

Creating a Security Policy Using the REST APIs **287**

Layer 4 to Layer 7 Parameters **288**

About Modifying the Configuration Parameters of a Deployed Service Graph	288
Example XML POST for an Application EPG With Configuration Parameters	288
Example XML of Configuration Parameters Inside the Device Package	289
Example XML POST for an Abstract Function Node With Configuration Parameters	290
Example XML POST for an Abstract Function Profile With Configuration Parameters	290
Copy Services	291
About Copy Services	291
Configuring Copy Services Using the REST API	291
Developing Automation	293
About the REST APIs	293
Examples of Automating Using the REST APIs	294
Example: Configuring Layer 4 to Layer 7 Services (Firewall)	300
Example: Configuring Layer 4 to Layer 7 Services Using the REST API	300
Example: Configuring Layer 4 to Layer 7 Route Peering	309
Configuring Layer 4 to Layer 7 Route Peering With the REST API	309
Specifying an l3extOut Policy for Layer 4 to L7 Route Peering	311

CHAPTER 15

Configuring QoS	313
CoS Preservation	313
Preserving 802.1P Class of Service Settings	313
Preserving QoS CoS Settings Using the REST API	314
Multipod QoS	315
Preserving QoS Priority Settings in a Multipod Fabric	315
Creating a DSCP Policy Using the REST API	316
Translating QoS Ingress Markings to Egress Markings	316
Translating QoS Ingress Markings to Egress Markings	316
Translating QoS Ingress Markings to Egress Markings Using the REST API	317
Troubleshooting Cisco APIC QoS Policies	317

CHAPTER 16

Configuring Security	319
Enabling TACACS+, RADIUS, and LDAP	319
Overview	319
Guidelines	319

Configuring APIC for TACACS+ Using the REST API	320
Configuring APIC for RADIUS Using the REST API	320
Configuring APIC for LDAP Using the REST API	321
Configuring FIPS	322
About Federal Information Processing Standards (FIPS)	322
Guidelines and Limitations	322
Configuring FIPS for Cisco APIC Using REST API	323
Configuring Fabric Secure Mode	323
Fabric Secure Mode	323
Configuring Fabric Secure Mode Using the REST API	324
Enabling RBAC	324
Access Rights Workflow Dependencies	324
User Access, Authorization, and Accounting	325
Multiple Tenant Support	325
User Access: Roles, Privileges, and Security Domains	325
AAA RBAC Roles and Privileges	326
Custom Roles	337
Sample RBAC Rules	337
Enabling Port Security	340
About Port Security and ACI	340
Port Security Guidelines and Restrictions	340
Port Security and Learning Behavior	340
Port Security at Port Level	341
Protect Mode	341
Configuring Port Security Using REST API	341
Enabling COOP Authentication	342
Overview	342
Using COOP with Cisco APIC	342
Guidelines and Limitations	343
Configuring COOP Authentication Using the REST API	343



Preface

This preface includes the following sections:

- [Audience, page xix](#)
- [New and Changed Information, page xix](#)
- [Document Conventions, page xxi](#)
- [Related Documentation, page xxiii](#)
- [Documentation Feedback, page xxiii](#)

Audience

This guide is intended primarily for data center administrators with responsibilities and expertise in one or more of the following:

- Virtual machine installation and administration
- Server administration
- Switch and network administration

New and Changed Information

The following table provides an overview of the significant changes to this guide up to this current release. The table does not provide an exhaustive list of all changes made to the guide or of the new features up to this release.

Table 1: New Features and Changed Information in this Document for Cisco APIC 2.2(1n) Release

Feature or Change	Description	Where Documented
Part 3: Setting Up APIC and the Fabric with the REST API	New section added.	Part 3

Feature or Change	Description	Where Documented
Managing Layer 4 to Layer 7 Services	Moved from Part 2 to Part 3	<i>Managing Layer 4 to Layer 7 Services</i> in <i>Setting up APIC and the Fabric with the REST API</i>
802.1Q Tunnels	You can now configure 802.1Q tunnels to enable point-to-multi-point tunneling of Ethernet frames in the fabric, with Quality of Service (QoS) priority settings.	<i>802.1Q Tunnels</i> in <i>Provisioning Layer 2 Networks</i>
APIC Cluster High Availability	Support is added to operate the APICs in a cluster in an Active/Standby mode. In an APIC cluster, the designated active APICs share the load and the designated standby APICs can act as an replacement for any of the APICs in an active cluster.	<i>Managing Cluster High Availability</i> in <i>Managing APIC Clusters</i>
Contract Preferred Groups	Support is added for contract preferred groups that enable greater control of communication between EPGs in a VRF. If most of the EPGs in the VRF should have open communication, but a few should only have limited communication with the other EPGs, you can configure a combination of a contract preferred group and contracts with filters to control communication precisely.	<i>Contract Preferred Groups</i> in <i>Configuring Tenants</i>
Dynamic Breakout Ports	Support is added for connecting a 40 Gigabit Ethernet (GE) leaf switch port to 4-10GE capable (downlink) devices (with Cisco 40-Gigabit to 4X10-Gigabit breakout cables).	<i>Dynamic Breakout Ports</i> in <i>Provisioning Layer 2 Networks</i>
FCoE over FEX Ports	You can now configure FCoE over FEX ports.	<i>Configuring FCoE over FEX Using the REST API</i> in <i>Provisioning Layer 2 Networks</i>
HSRP	Support is added for HSRP, a protocol that provides first-hop routing redundancy for IP hosts on Ethernet networks configured with a default router IP address.	<i>HSRP</i> in <i>Provisioning Layer 3 Outside Connections</i>

Feature or Change	Description	Where Documented
NetFlow	Support is added for NetFlow technology, which provides the metering base for a key set of applications, including network traffic accounting, usage-based network billing, network planning, as well as denial of services monitoring, network monitoring, outbound marketing, and data mining for both service providers and enterprise customers.	<i>NetFlow in Provisioning Core Services</i>

Table 2: New Features and Changed Information in this Document for Cisco APIC 2.1(1h) release

Feature or Change	Description	Where Documented
The document was created.		

Document Conventions

Command descriptions use the following conventions:

Convention	Description
bold	Bold text indicates the commands and keywords that you enter literally as shown.
<i>Italic</i>	Italic text indicates arguments for which the user supplies the values.
[x]	Square brackets enclose an optional element (keyword or argument).
[x y]	Square brackets enclosing keywords or arguments separated by a vertical bar indicate an optional choice.
{x y}	Braces enclosing keywords or arguments separated by a vertical bar indicate a required choice.
[x {y z}]	Nested set of square brackets or braces indicate optional or required choices within optional or required elements. Braces and a vertical bar within square brackets indicate a required choice within an optional element.
variable	Indicates a variable for which you supply values, in context where italics cannot be used.

Convention	Description
string	A nonquoted set of characters. Do not use quotation marks around the string or the string will include the quotation marks.

Examples use the following conventions:

Convention	Description
screen font	Terminal sessions and information the switch displays are in screen font.
boldface screen font	Information you must enter is in boldface screen font.
<i>italic screen font</i>	Arguments for which you supply values are in italic screen font.
< >	Nonprinting characters, such as passwords, are in angle brackets.
[]	Default responses to system prompts are in square brackets.
!, #	An exclamation point (!) or a pound sign (#) at the beginning of a line of code indicates a comment line.

This document uses the following conventions:


Note

Means *reader take note*. Notes contain helpful suggestions or references to material not covered in the manual.


Caution

Means *reader be careful*. In this situation, you might do something that could result in equipment damage or loss of data.


Warning
IMPORTANT SAFETY INSTRUCTIONS

This warning symbol means danger. You are in a situation that could cause bodily injury. Before you work on any equipment, be aware of the hazards involved with electrical circuitry and be familiar with standard practices for preventing accidents. Use the statement number provided at the end of each warning to locate its translation in the translated safety warnings that accompanied this device.

SAVE THESE INSTRUCTIONS

Related Documentation

Cisco Application Centric Infrastructure (ACI) Documentation

The ACI documentation is available at the following URL: <http://www.cisco.com/c/en/us/support/cloud-systems-management/application-policy-infrastructure-controller-apic/tsd-products-support-series-home.html>.

Cisco Application Centric Infrastructure (ACI) Simulator Documentation

The Cisco ACI Simulator documentation is available at <http://www.cisco.com/c/en/us/support/cloud-systems-management/application-centric-infrastructure-simulator/tsd-products-support-series-home.html>.

Cisco Nexus 9000 Series Switches Documentation

The Cisco Nexus 9000 Series Switches documentation is available at <http://www.cisco.com/c/en/us/support/switches/nexus-9000-series-switches/tsd-products-support-series-home.html>.

Cisco Application Virtual Switch Documentation

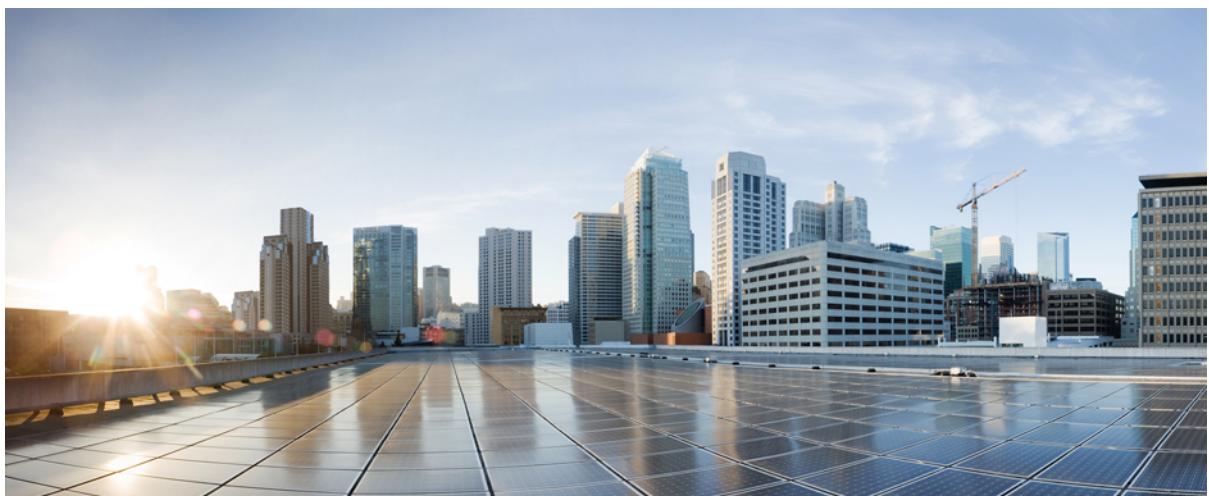
The Cisco Application Virtual Switch (AVS) documentation is available at <http://www.cisco.com/c/en/us/support/switches/application-virtual-switch/tsd-products-support-series-home.html>.

Cisco Application Centric Infrastructure (ACI) Integration with OpenStack Documentation

Cisco ACI integration with OpenStack documentation is available at <http://www.cisco.com/c/en/us/support/cloud-systems-management/application-policy-infrastructure-controller-apic/tsd-products-support-series-home.html>.

Documentation Feedback

To provide technical feedback on this document, or to report an error or omission, please send your comments to apic-docfeedback@cisco.com. We appreciate your feedback.



PART

Part 1: Cisco APIC REST API Usage Guidelines

- [Using the REST API, page 3](#)



CHAPTER 1

Using the REST API

- [About the REST API, page 3](#)
- [Composing REST API Requests, page 6](#)
- [Composing REST API Queries, page 15](#)
- [REST API Examples, page 23](#)
- [Accessing the REST API, page 30](#)
- [REST API Tools, page 38](#)

About the REST API

About the REST API

The Application Policy Infrastructure Controller (APIC) REST API is a programmatic interface that uses REST architecture. The API accepts and returns HTTP (not enabled by default) or HTTPS messages that contain JavaScript Object Notation (JSON) or Extensible Markup Language (XML) documents. You can use any programming language to generate the messages and the JSON or XML documents that contain the API methods or Managed Object (MO) descriptions.

The REST API is the interface into the management information tree (MIT) and allows manipulation of the object model state. The same REST interface is used by the APIC CLI, GUI, and SDK, so that whenever information is displayed, it is read through the REST API, and when configuration changes are made, they are written through the REST API. The REST API also provides an interface through which other information can be retrieved, including statistics, faults, and audit events. It even provides a means of subscribing to push-based event notification, so that when a change occurs in the MIT, an event can be sent through a web socket.

Standard REST methods are supported on the API, which includes POST, GET, and DELETE operations through HTTP. The POST and DELETE methods are idempotent, meaning that there is no additional effect if they are called more than once with the same input parameters. The GET method is nullipotent, meaning that it can be called zero or more times without making any changes (or that it is a read-only operation).

Payloads to and from the REST interface can be encapsulated through either XML or JSON encoding. In the case of XML, the encoding operation is simple: the element tag is the name of the package and class, and any

properties of that object are specified as attributes of that element. Containment is defined by creating child elements.

For JSON, encoding requires definition of certain entities to reflect the tree-based hierarchy; however, the definition is repeated at all levels of the tree, so it is fairly simple to implement after it is initially understood.

- All objects are described as JSON dictionaries, in which the key is the name of the package and class. The value is another nested dictionary with two keys: attribute and children.
- The attribute key contains a further nested dictionary describing key-value pairs that define attributes on the object.
- The children key contains a list that defines all the child objects. The children in this list are dictionaries containing any nested objects, which are defined as described here.

Authentication

REST API username- and password-based authentication uses a special subset of request Universal Resource Identifiers (URIs), including **aaaLogin**, **aaaLogout**, and **aaaRefresh** as the DN targets of a POST operation. Their payloads contain a simple XML or JSON payload containing the MO representation of an **aaaUser** object with the attribute **name** and **pwd** defining the username and password: for example, <**aaaUser name='admin' pwd='password'**>. The response to the POST operation will contain an authentication token as both a Set-Cookie header and an attribute to the **aaaLogin** object in the response named **token**, for which the XPath is **/imdata/aaaLogin/@token** if the encoding is XML. Subsequent operations on the REST API can use this token value as a cookie named **APIC-cookie** to authenticate future requests.

Subscription

The REST API supports the subscription to one or more MOs during your active API session. When any MO is created, changed, or deleted because of a user- or system-initiated action, an event is generated. If the event changes the data on any of the active subscribed queries, the APIC will send out a notification to the API client that created the subscription.

Management Information Model

All the physical and logical components that comprise the Application Centric Infrastructure fabric are represented in a hierarchical management information model (MIM), also referred to as the MIT. Each node in the tree represents an MO or group of objects that contains its administrative state and its operational state.

To view the MIM, see *Cisco APIC Management Information Model Reference Guide*.

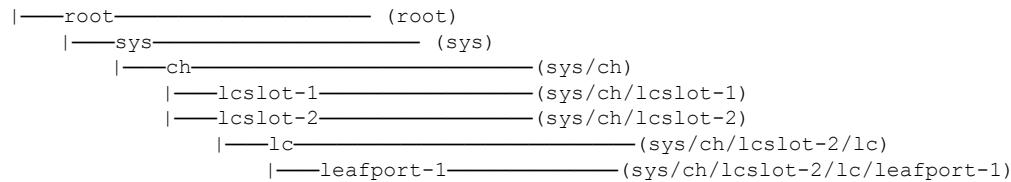
The hierarchical structure starts at the top (Root) and contains parent and child nodes. Each node in this tree is an MO and each object in the ACI fabric has a unique distinguished name (DN) that describes the object and its place in the tree. MOs are abstractions of the fabric resources. An MO can represent a physical object, such as a switch or adapter, or a logical object, such as a policy or fault.

Configuration policies make up the majority of the policies in the system and describe the configurations of different ACI fabric components. Policies determine how the system behaves under specific circumstances. Certain MOs are not created by users but are automatically created by the fabric (for example, power supply objects and fan objects). By invoking the API, you are reading and writing objects to the MIM.

The information model is centrally stored as a logical model by the APIC, while each switch node contains a complete copy as a concrete model. When a user creates a policy in the APIC that represents a configuration, the APIC updates the logical model. The APIC then performs the intermediate step of creating a fully elaborated policy from the user policy and then pushes the policy into all the switch nodes where the concrete model is

updated. The models are managed by multiple data management engine (DME) processes that run in the fabric. When a user or process initiates an administrative change to a fabric component (for example, when you apply a profile to a switch), the DME first applies that change to the information model and then applies the change to the actual managed endpoint. This approach is called a model-driven framework.

The following branch diagram of a leaf switch port starts at the top Root of the ACI fabric MIT and shows a hierarchy that comprises a chassis with two line module slots, with a line module in slot 2.



Object Naming

You can identify a specific object by its distinguished name (DN) or by its relative name (RN).



Note

You cannot rename an existing object. To simplify references to an object or group of objects, you can assign an alias or a tag.

Distinguished Name

The DN enables you to unambiguously identify a specific target object. The DN consists of a series of RNs:

dn = {rn}/{rn}/{rn}/{rn}...

In this example, the DN provides a fully qualified path for `fabport-1` from the top of the object tree to the object. The DN specifies the exact managed object on which the API call is operating.

```
< dn ="sys/ch/lcslot-1/lc/fabport-1" />
```

Relative Name

The RN identifies an object from its siblings within the context of its parent object. The DN contains a sequence of RNs.

For example, this DN:

```
<dn = "sys/ch/lcslot-1/lc/fabport-1"/>
```

contains these RNs:

Relative Name	Class	Description
sys	top:System	Top level of this system
ch	eqpt:Ch	Hardware chassis container
lcslot-1	eqpt:LCSlot	Line module slot 1

Relative Name	Class	Description
lc	eqpt:LC	Line (I/O) module
fabport-1	eqpt:FabP	Fabric-facing external I/O port 1

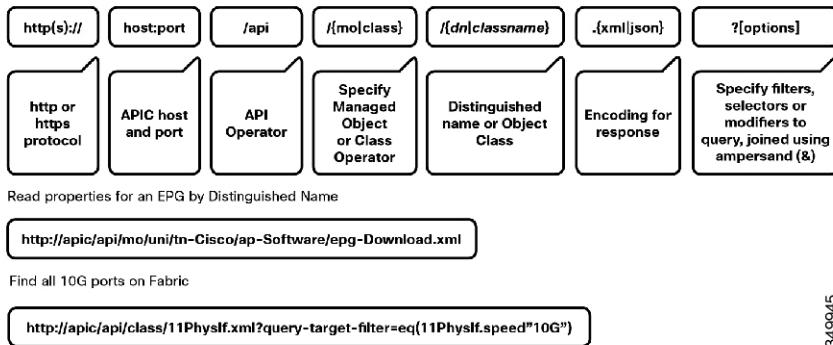
Composing REST API Requests

Read and Write Operations and Filters

Read Operations

After the object payloads are properly encoded as XML or JSON, they can be used in create, read, update, or delete operations on the REST API. The following diagram shows the syntax for a read operation from the REST API.

Figure 1: REST syntax



Because the REST API is HTTP-based, defining the URI to access a certain resource type is important. The first two sections of the request URI simply define the protocol and access details of the APIC. Next in the request URI is the literal string **/api**, indicating that the API will be invoked. Generally, read operations are for an object or class, as discussed earlier, so the next part of the URI specifies whether the operation will be for an MO or class. The next component defines either the fully qualified domain name (DN) being queried for object-based queries, or the package and class name for class-based queries. The final mandatory part of the request URI is the encoding format: either `.xml` or `.json`. This is the only method by which the payload format is defined. (The APIC ignores Content-Type and other headers.)

Write Operations

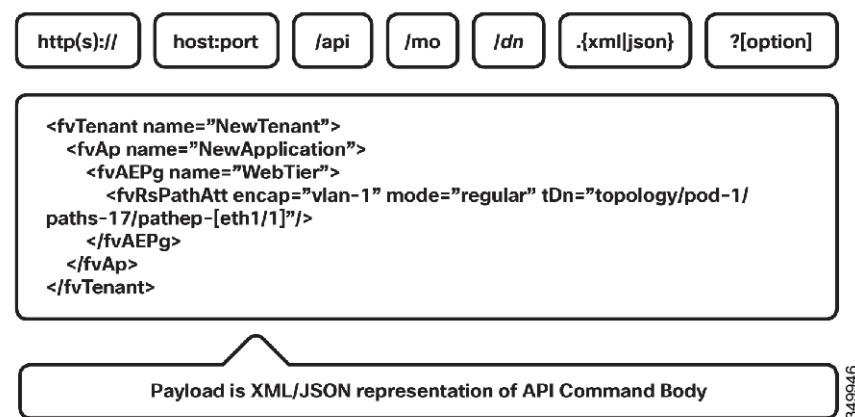
Both create and update operations in the REST API are implemented using the POST method, so that if an object does not already exist, it will be created, and if it does already exist, it will be updated to reflect any changes between its existing state and desired state.

Both create and update operations can contain complex object hierarchies, so that a complete tree can be defined in a single command so long as all objects are within the same context root and are under the 1MB

limit for data payloads for the REST API. This limit is in place to guarantee performance and protect the system under high loads.

The context root helps define a method by which the APIC distributes information to multiple controllers and helps ensure consistency. For the most part, the configuration should be transparent to the user, though very large configurations may need to be broken into smaller pieces if they result in a distributed transaction.

Figure 2: REST Payload



Create and update operations use the same syntax as read operations, except that they are always targeted at an object level, because you cannot make changes to every object of a specific class (nor would you want to). The create or update operation should target a specific managed object, so the literal string **/mo** indicates that the DN of the managed object will be provided, followed next by the actual DN. Filter strings can be applied to POST operations; if you want to retrieve the results of your POST operation in the response, for example, you can pass the **rsp-subtree=modified** query string to indicate that you want the response to include any objects that have been modified by your POST operation.

The payload of the POST operation will contain the XML or JSON encoded data representing the MO that defines the Cisco API command body.

Filters

The REST API supports a wide range of flexible filters, useful for narrowing the scope of your search to allow information to be located more quickly. The filters themselves are appended as query URI options, starting with a question mark (?) and concatenated with an ampersand (&). Multiple conditions can be joined together to form complex filters.

The following query filters are available:

Table 3: Query Filters

Filter Type	Syntax	Cobra Query Property	Description
query-target	{self children subtree}	AbstractQuery.queryTarget	Define the scope of a query

Filter Type	Syntax	Cobra Query Property	Description
target-subtree-class	<i>class name</i>	AbstractQuery.classFilter	Respond-only elements including the specified class
query-target-filter	<i>filter expressions</i>	AbstractQuery.propFilter	Respond-only elements matching conditions
rsp-subtree	{no children full}	AbstractQuery.subtree	Specifies child object level included in the response
rsp-subtree-class	<i>class name</i>	AbstractQuery.subtreeClassFilter	Respond only specified classes
rsp-subtree-filter	<i>filter expressions</i>	AbstractQuery.subtreePropFilter	Respond only classes matching conditions
rsp-subtree-include	{faults health :stats}	AbstractQuery.subtreeInclude	Request additional objects
order-by	<i>classname.property</i> {asc desc}	Not Implemented	Sort the response based on the property values

Using Classes in REST API Commands

The Application Policy Infrastructure Controller (APIC) classes are crucial from an operational perspective to understand how system events and faults relate to objects within the object model. Each event and/or fault in the system is a unique object that can be accessed for configuration, health, fault, and/or statistics.

All the physical and logical components that make up the Cisco Application Centric Infrastructure (ACI) fabric are represented in a hierarchical management information tree (MIT). Each node in the tree represents a managed object (MO) or group of objects that contains its administrative state and its operational state.

To access the complete list of classes, point to the APIC and reference the `doc/html` directory at the end of the URL:

```
https://apic-ip-address/doc/html/
```

Using Managed Objects in REST API Commands

Before performing an API operation on a managed object (MO) or its properties, you should view the object's class definition in the *Cisco APIC Management Information Model Reference*, which is a web-based document. The Management Information Model (MIM) serves as a schema that defines rules such as the following:

- The classes of parent objects to which an MO can be attached
- The classes of child objects that can be attached to an MO
- The number of child objects of a class type that can be attached to an MO
- Whether a user can create, modify, or delete an MO, and the privilege level required to do so

- The properties (attributes) of an object class
- The data type and range of a property

When you send an API command, the APIC checks the command for conformance with the MIM schema. If an API command violates the MIM schema, the APIC rejects the command and returns an error message. For example, you can create an MO only if it is allowed in the path you have specified in the command URI and only if you have the required privilege level for that object class. You can configure an MO's properties only with valid data, and you cannot create properties.

When composing an API command to create an MO, you need only include enough information in the command's URI and data structure to uniquely define the new MO. If you omit the configuration of a property when creating the MO, the property is populated with a default value if the MIM specifies one, or it is left blank.

When modifying a property of an MO, you need only specify the property to be modified and its new value. Other properties will be left unchanged.

Guidelines and Restrictions

- When you modify an MO that affects APIC or switch management communication policy, you might experience a brief disruption of any operations in progress on any APIC or switch web interface in the fabric. Configuration changes that can result in disruption include the following:
 - Changing management port settings, such as port number
 - Enabling or disabling HTTPS
 - Changing the state of redirection to HTTPS
 - Public key infrastructure (PKI) changes, such as key ring
- When you read an existing MO, any password property of the MO is read as blank for security reasons. If you then write the MO back to APIC, the password property is written as blank.



Tip

If you need to store an MO with its password information, use a configuration export policy. To store a specific MO, specify the MO as the target distinguished name in the policy.

Creating the API Command

You can invoke an API command or query by sending an HTTP or HTTPS message to the APIC with a URI of this form for an operation on a managed object (MO):

`{http| https}://host[:port]/api/mo/dn.{json| xml}[?options]`

Use this form for an operation on an object class:

`{http| https}://host[:port]/api/class/className.{json| xml}[?options]`

This example shows a URI for an API operation that involves an MO of class fv:Tenant:

```
https://apic-ip-address/api/mo/uni/tn-ExampleCorp.xml
```

URI Components

The components of the URI are as follows:

- `http://` or `https://`—Specifies HTTP or HTTPS. By default, only HTTPS is enabled. HTTP or HTTP-to-HTTPS redirection, if desired, must be explicitly enabled and configured, as described in [Configuring HTTP and HTTPS Using the GUI, on page 31](#). HTTP and HTTPS can coexist.
 - `host`—Specifies the hostname or IP address of the APIC.
 - `:port`—Specifies the port number for communicating with the APIC. If your system uses standard port numbers for HTTP (80) or HTTPS (443), you can omit this component.
 - `/api/`—Specifies that the message is directed to the API.
 - `mo | class`—Specifies whether the target of the operation is an MO or an object class.
 - `dn`—Specifies the distinguished name (DN) of the targeted MO.
 - `className`—Specifies the name of the targeted class. This name is a concatenation of the package name of the object queried and the name of the class queried in the context of the corresponding package.
- For example, the class aaa:User results in a `className` of aaaUser in the URI.
- `json | xml`—Specifies whether the encoding format of the command or response HTML body is JSON or XML.
 - `?options`—(Optional) Specifies one or more filters, selectors, or modifiers to a query. Multiple option statements are joined by an ampersand (&).

The URI for an API Operation on an MO

In an API operation to create, read, update, or delete a specific MO, the resource path consists of `/mo/` followed by the DN of the MO as described in the *Cisco APIC Management Information Model Reference*. For example, the DN of a tenant object, as described in the reference definition of class fv:Tenant, is `uni/tn-[name]`. This URI specifies an operation on an fv:Tenant object named ExampleCorp:

```
https://apic-ip-address/api/mo/uni/tn-ExampleCorp.xml
```

Alternatively, in a POST operation, you can POST to `/api/mo` and provide the DN in the body of the message, as in this example:

```
POST https://apic-ip-address/api/mo.xml
<fvTenant dn="uni/tn-ExampleCorp"/>
```

You can also provide only the name in the message body and POST to `/api/mo` and the remaining RN components, as in this example:

```
POST https://apic-ip-address/api/mo/uni.xml
<fvTenant name="ExampleCorp"/>
```

The URI for an API Operation on a Node MO

In an API operation to access an MO on a specific node device in the fabric, the resource path consists of `/mo/topology/pod-number/node-number/sys/` followed by the node component. For example, to access a board sensor in chassis slot b of node-1 in pod-1, use this URI:

```
GET https://apic-ip-address/api/mo/topology/pod-1/node-1/sys/ch/bslot/board/sensor-3.json
```

The URI for an API Operation on a Class

In an API operation to get information about a class, the resource path consists of `/class/` followed by the name of the class as described in the *Cisco APIC Management Information Model Reference*. In the URI, the colon in the class name is removed. For example, this URI specifies a query on the class aaaUser:

```
GET https://apic-ip-address/api/class/aaaUser.json
```

Composing the API Command Body

The HTML body of a POST operation must contain a JSON or XML data structure that provides the essential information necessary to execute the command. No data structure is sent with a GET or DELETE operation.

Guidelines for Composing the API Command Body

- The data structure does not need to represent the entire set of attributes and elements of the target MO or method, but it must provide at least the minimum set of properties or parameters necessary to identify the MO and to execute the command, not including properties or parameters that are incorporated into the URI.
- The data structure is a single tree in which all child nodes are unique with a unique DN. Duplicate nodes are not allowed. You cannot make two changes to a node by including the same node twice. In this case, you must merge your changes into a single node.
- In the data structure, the colon after the package name is omitted from class names and method names. For example, in the data structure for an MO of class zzz:Object, label the class element as zzzObject.
- Although the JSON specification allows unordered elements, the APIC REST API requires that the JSON 'attributes' element precede the 'children' array or other elements.
- If an XML data structure contains no children or subtrees, the object element can be self-closing.
- The API is case sensitive.
- When sending an API command, with 'api' in the URL, the maximum size of the HTML body for the API POST command is 1 MB.
- When uploading a device package file, with 'ppi' in the URL, the maximum size of the HTML body for the POST command is 10 MB.

Composing the API Command Body to Call a Method

To compose a command to call a method, create a JSON or XML data structure containing the parameters of the method using the method description in the *Cisco APIC Management Information Model Reference*.

The API reference for a typical method lists its input parameters, if any, and its return values, if any. The method is called with a structure containing the essential input parameters, and a successful response returns a complete structure containing the return values.

The description for a hypothetical method config:Method might appear in the API reference as follows:

```
Method config:Method(
    inParameter1,
    inParameter2,
    inParameter3,
    outParameter1,
    outParameter2
)
```

The parameters beginning with "in" represent the input parameters. The parameters beginning with "out" represent values returned by the method. Parameters with no "in" or "out" prefix are input parameters.

A JSON structure to call the method resembles the following structure:

```
{
  "configMethod": {
    "attributes": {
      "inParameter1": "value1",
      "inParameter2": "value2",
      "inParameter3": "value3"
    }
  }
}
```

An XML structure to call the method resembles the following structure:

```
<configMethod
    inParameter1="value1"
    inParameter2="value2"
    inParameter3="value3"
/>
```



The parameters of some methods include a substructure, such as filter settings or configuration settings for an MO. For specific information, see the method description in the *Cisco APIC Management Information Model Reference*.

Composing the API Command Body for an API Operation on an MO

To compose a command to create, modify, or delete an MO, create a JSON or XML data structure that describes the essential properties and children of the object's class by using the class description in the *Cisco APIC Management Information Model Reference*. You can omit any attributes or children that are not essential to execute the command.

A JSON structure for an MO of hypothetical class zzz:Object resembles this structure:

```
{
  "zzzObject" : {
    "attributes" : {
      "property1" : "value1",
      "property2" : "value2",
      "property3" : "value3"
    },
  }
}
```

```

    "children" :
    [
        {
            "zzzChild1" : {
                "attributes" : {
                    "childProperty1" : "childValue1",
                    "childProperty2" : "childValue1"
                },
                "children" : []
            }
        }
    ]
}

```

An XML structure for an MO of hypothetical class zzz:Object resembles this structure:

```

<zzzObject
    property1 = "value1",
    property2 = "value2",
    property3 = "value3">
    <zzzChild1
        childProperty1 = "childValue1",
        childProperty2 = "childValue1">
    </zzzChild1>
</zzzObject>

```

A successful operation returns a complete data structure for the MO.

Using Tags and Alias

To simplify API operations, you can assign tags or an alias to an object. In an API operation, you can refer to the object or group of objects by the alias or tag name instead of by the distinguished name (DN). Tags and aliases differ in their usage as follows:

- Tag—A tag allows you to group multiple objects by a descriptive name. You can assign the same tag name to multiple objects and you can assign one or more tag names to an object.
- Alias—An alias can be a simpler and more descriptive name than the DN when referring to a single object. You can assign a particular alias name to only one object. The system will prevent you from assigning the same alias name to a second object.



Note

Not every object supports a tag. To determine whether an object is taggable, inspect the class of the object in the *Cisco APIC Management Information Model Reference*. If the contained hierarchy of the object class includes a tag instance (such as `tag:AInst` or a class that derives from `tag:AInst`), an object of that class can be tagged.

Adding Tags

You can add one or more tags by using the following syntax in the URI of an API POST operation:

`/api/tag/mo/dn.{json|xml}?add=[, name, ...][, name, ...]`

In this syntax, `name` is the name of a tag and `dn` is the distinguished name of the object to which the tag is assigned.

This example shows how to assign the tags tenants and orgs to the tenant named ExampleCorp:

```
POST https://apic-ip-address/api/tag/mo/uni/tn-ExampleCorp.xml?add=tenants,orgs
```

Removing Tags

You can remove one or more tags by using the following syntax in the URI of an API POST operation:

```
/api/tag/mo/dn.{json|xml}?remove=name[ , name, ... ]
```

This example shows how to remove the tag orgs from the tenant named ExampleCorp:

```
POST https://apic-ip-address/api/tag/mo/uni/tn-ExampleCorp.xml?remove=orgs
```

You can delete all instances of a tag by using the following syntax in the URI of an API DELETE operation:

```
/api//tag/name.{json|xml}
```

This example shows how to remove the tag orgs from all objects:

```
DELETE https://apic-ip-address/api/tag/orgs.xml
```

Adding an Alias

You can add an alias by using the following syntax in the URI of an API POST operation:

```
/api/alias/mo/dn.{json|xml}?set=name
```

In this syntax, *name* is the name of the alias and *dn* is the distinguished name of the object to which the alias is assigned.

This example shows how to assign the alias tenant8 to the tenant named ExampleCorp:

```
POST https://apic-ip-address/api/alias/mo/uni/tn-ExampleCorp.xml?set=tenant8
```

Removing an Alias

You can remove an alias by using the following syntax in the URI of an API POST operation:

```
/api/alias/mo/dn.{json|xml}?clear=yes
```

This example shows how to remove any alias from the tenant named ExampleCorp:

```
POST https://apic-ip-address/api/alias/mo/uni/tn-ExampleCorp.xml?clear=yes
```

Additional Examples



Note In the examples in this section, the responses have been edited to remove attributes unrelated to tags.

This example shows how to find all tags assigned to the tenant named ExampleCorp:

```
GET https://apic-ip-address/api/tag/mo/uni/tn-ExampleCorp.xml

RESPONSE:
<imdata>
  <tagInst
    dn="uni/tn-ExampleCorp/tag-tenants"
    name="tenants"
  />
  <tagInst
    dn="uni/tn-ExampleCorp/tag-orgs"
    name="orgs"
  />
</imdata>
```

This example shows how to find all objects with the tag 'tenants':

```
GET https://apic-ip-address/api/tag/tenants.xml

RESPONSE:
<imdata>
  <fvTenant
    dn="uni/tn-ExampleCorp"
    name="ExampleCorp"
  />
</imdata>
```

Composing REST API Queries

Composing Query Filter Expressions

You can filter the response to an API query by applying an expression of logical operators and values.

A basic equality or inequality test is expressed as follows:

```
query-target-filter=[eq|ne] (attribute,value)
```

You can create a more complex test by combining operators and conditions using parentheses and commas:

```
query-target-filter=[and|or] ([eq|ne] (attribute,value),[eq|ne] (attribute,value),...)
```



Note

A scoping filter can contain a maximum of 20 '(attribute,value)' filter expressions. If the limit is exceeded, the API returns an error.

Available Logical Operators

This table lists the available logical operators for query filter expressions.

Operator	Description
eq	Equal to
ne	Not equal to

Operator	Description
lt	Less than
gt	Greater than
le	Less than or equal to
ge	Greater than or equal to
bw	Between
not	Logical inverse
and	Logical AND
or	Logical OR
xor	Logical exclusive OR
true	Boolean TRUE
false	Boolean FALSE
anybit	TRUE if at least one bit is set
allbits	TRUE if all bits are set
wcard	Wildcard
pholder	Property holder
passive	Passive holder

Examples

This example returns all managed objects of class aaaUser whose last name is equal to "Washington":

```
GET https://apic-ip-address/api/class/aaaUser.json?
query-target-filter=eq(aaaUser.lastName,"Washington")
```

This example returns endpoint groups whose fabEncap property is "vxlan-12780288":

```
GET https://apic-ip-address/api/class/fvAEPg.xml?
query-target-filter=eq(fvAEPg.fabEncap,"vxlan-12780288")
```

This example shows all tenant objects with a current health score of less than 50:

```
GET https://apic-ip-address/api/class/fvTenant.json?
rsp-subtree/include=health,required
&
```

```
rsp-subtree-filter=lt(healthInst.cur,"50")
```

This example returns all endpoint groups and their faults under the tenant ExampleCorp:

```
GET https://apic-ip-address/api/mo/uni/tn-ExampleCorp.xml?
query-target=subtree
&
target-subtree-class=fvAEPg
&
rsp-subtree/include=faults
```

This example returns aaa:Domain objects whose names are not "infra" or "common":

```
GET https://apic-ip-address/api/class/aaaDomain.json?
query-target-filter=
and(ne(aaaDomain.name,"infra"),
ne(aaaDomain.name,"common"))
```

Applying Query Scoping Filters

You can limit the scope of the response to an API query by applying scoping filters. You can limit the scope to the first level of an object or to one or more of its subtrees or children based on the class, properties, categories, or qualification by a logical filter expression.

query-target={self | children | subtree}

This statement restricts the scope of the query. This list describes the available scopes:

- **self**—(Default) Considers only the MO itself, not the children or subtrees.
- **children**—Considers only the children of the MO, not the MO itself.
- **subtree**—Considers the MO itself and its subtrees.

target-subtree-class=mo-class1[,mo-class2]...

This statement specifies which object classes are to be considered when the **query-target** option is used with a scope other than **self**. You can specify multiple desired object types as a comma-separated list with no spaces.

To request subtree information, combine **query-target=subtree** with the **target-subtree-class** statement to indicate the specific subtree as follows:

```
query-target=subtree&target-subtree-class=className
```

This example requests information about the running firmware. The information is contained in the firmware:CtrlrRunning subtree (child) object of the APIC firmware status container firmware:CtrlrFwStatusCont:

```
GET https://apic-ip-address/api/class/firmwareCtrlrFwStatusCont.json?
query-target=subtree&target-subtree-class=firmwareCtrlrRunning
```

query-target-filter=filter-expression

This statement specifies a logical filter to be applied to the response. This statement can be used by itself or applied after the **query-target** statement.

rsp-subtree={no | children | full}

For objects returned, this option specifies whether child objects are included in the response. This list describes the available choices:

- **no**—(Default) Response includes no children.
- **children**—Response includes only the children.
- **full**—Response includes the entire structure including the children.

rsp-subtree-class=mo-class

When child objects are to be returned, this statement specifies that only child objects of the specified object class are included in the response.

rsp-subtree-filter=filter-expression

When child objects are to be returned, this statement specifies a logical filter to be applied to the child objects.

**Note**

When an **rsp-subtree-filter** query statement includes a *class.property* operand, the specified class name is used only to identify the property and its type. The returned results are not filtered by class, and may include any child object that contains a property of the same name but belonging to a different class if that object's property matches the query condition. To filter by class, you must use additional query filters.

rsp-subtree-include=category1[,category2...][option]

When child objects are to be returned, this statement specifies additional contained objects or options to be included in the response. You can specify one or more of the following categories in a comma-separated list with no spaces:

- **audit-logs**—Response includes subtrees with the history of user modifications to managed objects.
- **event-logs**—Response includes subtrees with event history information.
- **faults**—Response includes subtrees with currently active faults.
- **fault-records**—Response includes subtrees with fault history information.
- **health**—Response includes subtrees with current health information.
- **health-records**—Response includes subtrees with health history information.
- **relations**—Response includes relations-related subtree information.
- **stats**—Response includes statistics-related subtree information.
- **tasks**—Response includes task-related subtree information.

With any of the preceding categories, you can also specify one of the following options to further refine the query results:

- **count**—Response includes a count of matching subtrees but not the subtrees themselves.
- **no-scoped**—Response includes only the requested subtree information. Other top-level information of the target MO is not included in the response.

- **required**—Response includes only the managed objects that have subtrees matching the specified category.

For example, to include fault-related subtrees, specify **faults** in the list. To return only fault-related subtrees and no other top-level MO information, specify **faults,no-scoped** in the list as shown in this example:

```
query-target=subtree&rsp-subtree/include=faults,no-scoped
```



Note

Some types of child objects are not created until the parent object has been pushed to a fabric node (leaf). Until such a parent object has been pushed to a fabric node, a query on the parent object using the **rsp-subtree-include** filter might return no results. For example, a class query for `fvAEPg` that includes the query option `rsp-subtree-include=stats` will return stats only for endpoint groups that have been applied to a tenant and pushed to a fabric node.

rsp-prop-include={all | naming-only | config-only}

This statement specifies what type of properties should be included in the response when the **rsp-subtree** option is used with an argument other than **no**.

- **all**—Response includes all properties of the returned managed objects.
- **naming-only**—Response includes only the naming properties of the returned managed objects.
- **config-only**—Response includes only the configurable properties of the returned managed objects.



Note

If the managed object is not configurable or cannot be exported (backed up), the managed object is not returned.

Related Topics

[Composing Query Filter Expressions, on page 15](#)

[Example: Using the JSON API to Get Running Firmware, on page 27](#)

Filtering API Query Results

You can filter the results of an API query by appending one or more condition statements to the query URI as a parameter in this format:

```
https://URI?condition[&condition[&...]]
```

Multiple condition statements are joined by an ampersand (&).



Note

The condition statement must not contain spaces.

Options are available to filter by object attributes and object subtrees.

Filter Conditional Operators

The query filtering feature supports the following condition operators:

Operator	Description
eq	Equal to
ne	Not equal to
lt	Less than
gt	Greater than
le	Less than or equal to
ge	Greater than or equal to
bw	Between
not	Logical inverse
and	Logical AND
or	Logical OR
xor	Logical exclusive OR
true	Boolean TRUE
false	Boolean FALSE
anybit	TRUE if at least one bit is set
allbits	TRUE if all bits are set
wcard	Wildcard
pholder	Property holder
passive	Passive holder

Sorting and Paginating Query Results

When sending an API query that returns a large quantity of data, you can have the return data sorted and paginated to make it easier to find the information you need.

Sorting the Results

By adding the `order-by` operator to the query URI, you can sort the query response by one or more properties of a class, and you can specify the direction of the order using the following syntax.

`order-by=classname.property[| {asc| desc}][,classname.property[| {asc| desc}]]...]`

Use the optional pipe delimiter ('|') to specify either ascending order (`asc`) or descending order (`desc`). If no order is specified, the default is ascending order.

You can perform a multi-level sort by more than one property (for example, last name and first name), but all properties must be of the same MO or they must be inherited from the same abstract class.

This example shows you how to sort users by last name, then by first name:

```
GET  
https://apic-ip-address/api/class/aaaUser.json?order-by=aaaUser.lastName|asc,aaaUser.firstName|asc
```

Paginating the Results

By adding the `page-size` operator to the query URI, you can divide the query results into groups (pages) of objects using the following syntax. The operand specifies the number of objects in each group.

`page-size=number-of-objects-per-page`

By adding the `page` operator in the query URI, you can specify a single group to be returned using the following syntax. The pages start from number 0.

`page=page-number`

This example shows you how to specify 15 fault instances per page in descending order, returning only the first page:

```
GET  
https://apic-ip-address/api/class/faultInfo.json?order-by=faultInst.severity|desc&page=0&page-size=15
```



Note

Every query, whether paged or not, generates a new set of results. When you perform a query that returns only a single page, the query response includes a count of the total results, but the unsent pages are not stored and cannot be retrieved by a subsequent query. A subsequent query generates a new set of results and returns the page requested in that query.

Subscribing to Query Results

When you perform an API query, you have the option to create a subscription to any future changes in the results of that query that occur during your active API session. When any MO is created, changed, or deleted because of a user- or system-initiated action, an event is generated. If that event changes the results of an active subscribed query, the APIC generates a push notification to the API client that created the subscription.

Opening a WebSocket

The API subscription feature uses the WebSocket protocol (RFC 6455) to implement a two-way connection with the API client through which the API can send unsolicited notification messages to the client. To establish this notification channel, you must first open a WebSocket connection with the API. Only a single WebSocket connection is needed to support multiple query subscriptions with multiple APIC instances. The WebSocket connection is dependent on your API session connection, and closes when your API session ends.

The WebSocket connection is typically opened by a JavaScript method in an HTML5-compliant browser, as in the following example:

```
var Socket = new WebSocket('https://apic-ip-address/socket%TOKEN%');
```

In the URI, the **%TOKEN%** is the current API session token (cookie). This example shows the URI with a token:

```
https://apic-ip-address/socketGkZl5NLRZJ15+jqChouaZ9CYjgE58W/pMccR+LeXmdO0obG9NB  
Iwo1VBo7+YC1oiJL9mS6I9qh62BkX+Xddhe0JYrTmSG4JcKZ4t3bcP2Mxy3VBmgoJjwZ76Zouf9V9AD6X  
183lyoR4bLBzqbSSU1R2NIgUotCGWjZt5JX6CJF0=
```

After the WebSocket connection is established, it is not necessary to resend the API session token when the API session is refreshed.

Creating a Subscription

To create a subscription to a query, perform the query with the option `?subscription=yes`. This example creates a subscription to a query of the fv:Tenant class in the JSON format:

```
GET https://apic-ip-address/api/class/fvTenant.json?subscription=yes
```

The query response contains a subscription identifier, **subscriptionId**, that you can use to refresh the subscription and to identify future notifications from this subscription.

```
{
  "subscriptionId" : "72057611234574337",
  "imdata" : [
    {
      "fvTenant" : {
        "attributes" : {
          "instanceId" : "0:0",
          "childAction" : "",
          "dn" : "uni/tn-common",
          "lcOwn" : "local",
          "monPolDn" : "",
          "name" : "common",
          "replTs" : "never",
          "status" : ""
        }
      }
    }
  ]
}
```

Receiving Notifications

An event notification from the subscription delivers a data structure that contains the subscription ID and the MO description. In this JSON example, a new user has been created with the name "sysadmin5":

```
{
  "subscriptionId" : ["72057598349672454", "72057598349672456"],
  "imdata" : [
    {
      "aaaUser" : {

```

```

    "attributes" : {
        "accountStatus" : "active",
        "childAction" : "",
        "clearPwdHistory" : "no",
        "descr" : "",
        "dn" : "uni/userext/user-sysadmin5",
        "email" : "",
        "encPwd" : "TUxISkhH$VHyidGgBX0r7N/srt/YcMYTEn5248ommFhNFzZghvAU=",
        "expiration" : "never",
        "expires" : "no",
        "firstName" : "",
        "intId" : "none",
        "lastName" : "",
        "lcOwn" : "local",
        "name" : "sysadmin5",
        "phone" : "",
        "pwd" : "",
        "pwdLifeTime" : "no-password-expire",
        "pwdSet" : "yes",
        "rep1Ts" : "2013-05-30T11:28:33.835",
        "rn" : "",
        "status" : "created"
    }
}
]
}
}

```

Because multiple active subscriptions can exist for a query, a notification can contain multiple subscription IDs as in the example shown.



Note

Notifications are supported in either JSON or XML format.

Refreshing the Subscription

To continue to receive event notifications, you must periodically refresh each subscription during your API session. To refresh a subscription, send an HTTP GET message to the API method **subscriptionRefresh** with the parameter **id** equal to the **subscriptionId** as in this example:

```
GET https://apic-ip-address/api/subscriptionRefresh.json?id=72057611234574337
```

The API returns an empty response to the refresh message unless the subscription has expired.



Note

The timeout period for a subscription is one minute. To prevent lost notifications, you must send a subscription refresh message at least once every 60 seconds.

REST API Examples

Information About the API Examples

In the examples, the JSON and XML structures have been expanded with line feeds, spaces, and indentations for readability.

Example: Using the JSON API to Add a Leaf Port Selector Profile

This example shows how to add a leaf port selector profile.

As shown in the *Cisco APIC Management Information Model Reference*, this hierarchy of classes forms a leaf port selector profile:

- fabric:LePortP — A leaf port profile is represented by a managed object (MO) of this class, which has a distinguished name (DN) format of `uni/fabric/leportp-[name]`, in which `leportp-[name]` is the relative name (RN). The leaf port profile object is a template that can contain a leaf port selector as a child object.
 - fabric:LFPoS — A leaf port selector is represented by an MO of this class, which has a RN format of `leafports-[name]-typ-[type]`. The leaf port selector object can contain one or more ports or ranges of ports as child objects.
 - fabric:PortBlk — A leaf port or a range of leaf ports is represented by an MO of this class, which has a RN format of `portblk-[name]`.

The API command that creates the new leaf port selector profile MO can also create and configure the child MOs.

This example creates a leaf port selector profile with the name "MyLPSelectorProf." The example profile contains a selector named "MySelectorName" that selects leaf port 1 on leaf switch 1 and leaf ports 3 through 5 on leaf switch 1. To create and configure the new profile, send this HTTP POST message:

```
POST http://apic-ip-address/api/mo/uni/fabric/leportp-MyLPSelectorProf.json
{
  "fabricLePortP" : {
    "attributes" : {
      "descr" : "Selects leaf ports 1/1 and 1/3-5"
    },
    "children" : [
      {
        "fabricLFPoS" : {
          "attributes" : {
            "name" : "MySelectorName",
            "type" : "range"
          },
          "children" : [
            {
              "fabricPortBlk" : {
                "attributes" : {
                  "fromCard" : "1",
                  "toCard" : "1",
                  "fromPort" : "1",
                  "toPort" : "1",
                  "name" : "block2"
                }
              }
            },
            {
              "fabricPortBlk" : {
                "attributes" : {
                  "fromCard" : "1",
                  "toCard" : "1",
                  "fromPort" : "3",
                  "toPort" : "5",
                  "name" : "block3"
                }
              }
            }
          ]
        }
      }
    ]
  }
}
```

```

        }
    ]
}
}
```

A successful operation returns this response body:

```
{
  "imdata": [
    {
      "fabricLePortP": {
        "attributes": {
          "instanceId": "0:0",
          "childAction": "deleteNonPresent",
          "descr": "Select leaf ports 1/1 and 1/3-5",
          "dn": "uni/fabric/leportp-MyLPSelectorProf",
          "lcOwn": "local",
          "name": "MyLPSelectorProf",
          "replTs": "never",
          "rn": "",
          "status": "created"
        },
        "children": [
          {
            "fabricLFPorts": {
              "attributes": {
                "instanceId": "0:0",
                "childAction": "deleteNonPresent",
                "dn": "",
                "lcOwn": "local",
                "name": "MySelectorName",
                "replTs": "never",
                "rn": "lefabports-MySelectorName-typ-range",
                "status": "created",
                "type": "range"
              },
              "children": [
                {
                  "fabricPortBlk": {
                    "attributes": {
                      "instanceId": "0:0",
                      "childAction": "deleteNonPresent",
                      "dn": "",
                      "fromCard": "1",
                      "fromPort": "3",
                      "lcOwn": "local",
                      "name": "block3",
                      "replTs": "never",
                      "rn": "portblk-block3",
                      "status": "created",
                      "toCard": "1",
                      "toPort": "5"
                    }
                  }
                },
                {
                  "fabricPortBlk": {
                    "attributes": {
                      "instanceId": "0:0",
                      "childAction": "deleteNonPresent",
                      "dn": "",
                      "fromCard": "1",
                      "fromPort": "1",
                      "lcOwn": "local",
                      "name": "block2",
                      "replTs": "never",
                      "rn": "portblk-block2",
                      "status": "created",
                      "toCard": "1",
                      "toPort": "1"
                    }
                  }
                }
              ]
            }
          }
        ]
      }
    }
}
```

```

        ]
    }
]
}
}
```

To delete the new profile, send an HTTP POST message with a fabricLePortP attribute of "status":"deleted", as in this example:

```
POST http://apic-ip-address/api/mo/uni/fabric/leportp-MyLPSelectorProf.json

{
    "fabricLePortP" : {
        "attributes" : {
            "status" : "deleted"
        }
    }
}
```

Alternatively, you can send this HTTP DELETE message:

```
DELETE http://apic-ip-address/api/mo/uni/fabric/leportp-MyLPSelectorProf.json
```

Example: Using the JSON API to Get Information About a Node

This example shows how to query the APIC to access a node in the system.

To direct an API operation to a specific node device in the fabric, the resource path consists of /mo/topology/pod-number/node-number/sys/ followed by the node component. For example, this URI accesses board sensor 3 in chassis slot B of node 1:

```
GET http://apic-ip-address/api/mo/topology/pod-1/node-1/sys/ch/bslot/board/sensor-3.json
```

A successful operation returns a response body similar to this example:

```
{
    "imdata" :
    [
        {
            "eqptSensor" : {
                "attributes" : {
                    "instanceId" : "0:0",
                    "childAction" : "",
                    "dn" : "topology/pod-1/node-1/sys/ch/bslot/board/sensor-3",
                    "id" : "3",
                    "majorThresh" : "0",
                    "mfgTm" : "not-applicable",
                    "minorThresh" : "0",
                    "model" : "",
                    "monPolDn" : "",
                    "rev" : "0",
                    "ser" : "",
                    "status" : "",
                    "type" : "dimm",
                    "vendor" : "Cisco Systems, Inc."
                }
            }
        }
    ]
}
```

Example: Using the JSON API to Get Running Firmware

This example shows how to query the APIC to determine which firmware images are running.

The detailed information on running firmware is contained in an object of class firmware:CtrlrRunning, which is a child class (subtree) of the APIC firmware status container class firmware:CtrlrFwStatusCont. Because there can be multiple running firmware instances (one per APIC instance), you can query the container class and filter the response for the subtree of running firmware objects.

This example shows the API query message:

```
GET http://apic-ip-address/api/class/firmware:CtrlrFwStatusCont.json?
query-target=subtree
&
target-subtree-class=firmwareCtrlrRunning
```

A successful operation returns a response body similar to this example:

```
{
  "imdata": [
    {
      "firmwareCtrlrRunning": {
        "attributes": {
          "instanceId": "0:0",
          "applId": "3",
          "childAction": "",
          "dn": "Ctrlrfwstatuscont/ctrlrrunning-3",
          "lcOwn": "local",
          "replTs": "never",
          "rn": "",
          "status": "",
          "ts": "2012-12-31T16:00:00.000",
          "type": "ifc",
          "version": "1.1"
        }
      }
    },
    {
      "firmwareCtrlrRunning": {
        "attributes": {
          "instanceId": "0:0",
          "applId": "1",
          "childAction": "",
          "dn": "ctrlrfwstatuscont/ctrlrrunning-1",
          "lcOwn": "local",
          "replTs": "never",
          "rn": "",
          "status": "",
          "ts": "2012-12-31T16:00:00.000",
          "type": "ifc",
          "version": "1.1"
        }
      }
    },
    {
      "firmwareCtrlrRunning": {
        "attributes": {
          "instanceId": "0:0",
          "applId": "2",
          "childAction": "",
          "dn": "ctrlrfwstatuscont/ctrlrrunning-2",
          "lcOwn": "local",
          "replTs": "never",
          "rn": "",
          "status": "",
          "ts": "2012-12-31T16:00:00.000",
          "type": "ifc",
          "version": "1.1"
        }
      }
    }
  ]
}
```

```

        ]
    }

```

This response describes three running instances of APIC firmware version 1.1.

Example: Using the JSON API to Get Top Level System Elements

This example shows how to query the APIC to determine what system devices are present.

General information about the system elements (APICs, spines, and leafs) is contained in an object of class top:System.

This example shows the API query message:

```
GET http://apic-ip-address/api/class/topSystem.json
```

A successful operation returns a response body similar to this example:

```
{
  "imdata" :
  [
    {
      "topSystem" : {
        "attributes" : {
          "instanceId" : "0:0",
          "address" : "10.0.0.32",
          "childAction" : "",
          "currentTime" : "2013-06-14T04:13:05.584",
          "currentTimeZone" : "",
          "dn" : "topology/pod-1/node-17/sys",
          "fabricId" : "0",
          "id" : "17",
          "inbMgmtAddr" : "0.0.0.0",
          "lcOwn" : "local",
          "mode" : "unspecified",
          "name" : "leaf0",
          "nodeId" : "0",
          "oobMgmtAddr" : "0.0.0.0",
          "podId" : "1",
          "replTs" : "never",
          "role" : "leaf",
          "serial" : "FOX-270308",
          "status" : "",
          "systemUpTime" : "00:00:02:03"
        }
      }
    },
    {
      "topSystem" : {
        "attributes" : {
          "instanceId" : "0:0",
          "address" : "10.0.0.1",
          "childAction" : "",
          "currentTime" : "2013-06-14T04:13:29.301",
          "currentTimeZone" : "PDT",
          "dn" : "topology/pod-1/node-1/sys",
          "fabricId" : "0",
          "id" : "1",
          "inbMgmtAddr" : "0.0.0.0",
          "lcOwn" : "local",
          "mode" : "unspecified",
          "name" : "apic0",
          "nodeId" : "0",
          "oobMgmtAddr" : "0.0.0.0",
          "podId" : "0",
          "replTs" : "never",
          "role" : "apic",
          "serial" : "",
          "status" : "",
          "systemUpTime" : "00:00:02:26"
        }
      }
    }
  ]
}
```

```

    }
}
}
}
```

This response indicates that the system consists of one APIC (node-1) and one leaf (node-17).

Example: Using the XML API and OwnerTag to Add Audit Log Information to Actions

This example shows how to use the **ownerTag** or **ownerKey** property to add custom audit log information when an object is created or modified.

All configurable objects contain the properties **ownerTag** and **ownerKey**, which are user-configurable string properties. When any configurable object is created or modified by a user action, an audit log record object (**aaa:ModLR**) is automatically created to contain information about the change to be reported in the audit log. The audit log record object includes a list (the **changeSet** property) of the configured object's properties that were changed by the action. In the command to create or modify the configurable object, you can add your own specific tracking information, such as a job ticket number or the name of the person making the change, to the **ownerTag** or **ownerKey** property of the configurable object. This tracking information will then be included in the audit log record along with the details of the change.



Note

The **ownerTag** information will appear in the log only when the **ownerTag** contents have been changed. To include the same information in a subsequent configuration change, you can clear the **ownerTag** contents before making the next configuration change. This condition applies also to the **ownerKey** property.

In the following example, a domain reference is added to a tenant configuration. As part of the command, the operator's name is entered as the **ownerKey** and a job number is entered as the **ownerTag**.

```
POST https://apic-ip-address/api/mo/uni/tn-ExampleCorp.xml

<fvTenant name="ExampleCorp" ownerKey="georgewa" ownerTag="chg:00033">
    <aaaDomainRef name="ExampleDomain" ownerKey="georgewa" ownerTag="chg:00033"/>
</fvTenant>
```

In this case, two **aaa:ModLR** records are generated — one for the **fv:Tenant** object and one for the **aaa:DomainRef** object. Unless the **ownerKey** or **ownerTag** properties are unchanged from a previous configuration, their new values will appear in the **changeSet** list of the **aaa:ModLR** records, and this information will appear in the audit log record that reports this configuration change.

Example: XML Get Endpoints (Devices) with IP and MAC Addresses

The fvCEp class can be used to derive a list of endpoints (devices) attached to the fabric and the associated IP and MAC address and the encapsulation for each object.

Procedure

Use an XML query, such as the following example, to get a list of endpoints with the IP and MAC address for each one:

Example:

```
GET https://apic-ip-address/api/node/class/fvCEp.xml
```

Example: Monitoring Using the REST API

In the examples in this topic, the JSON and XML structures have been expanded with line feeds, spaces, and indentations for readability.

XML Example: Get the Current List of Faults in the Fabric

You can use the **faultInst** class to derive all faults associated with the fabric, tenant, or individual managed objects within the APIC. Send a query with XML such as this example:

```
GET https://apic-ip-address/api/node/class/faultInst.xml?  
query-target-filter=and(eq(faultInst.cause,"config-failure"))
```

XML Example: Get the Current List of Faults in the Fabric That Were Caused by a Failed Configuration

You can also use the **fault Inst** class with filters to limit the response to faults that were caused by a failed configuration, with XML such as this example:

```
GET https://apic-ip-address/api/node/class/faultInst.xml?  
query-target-filter=and(e(stultification,"config-failure"))
```

XML Example: Get the Properties for a Specific Managed Object, DN

You can use a MO query to obtain the properties of the tenant name, with XML such as the following example:

```
GET https://apic-ip-address/api/node/mo/uni/tn-common.xml?query-target=self
```

Accessing the REST API

Accessing the REST API

Procedure

By using a script or a browser-based REST client, you can send an API POST or GET message of the form:

https://apic-ip-address/api/api-message-url

Use the out-of-band management IP address that you configured during the initial setup.

Note

- Only https is enabled by default. By default, http and http-to-https redirection are disabled.
- You must send an authentication message to initiate an API session. Use the administrator login name and password that you configured during the initial setup.

Invoking the API

You can invoke an API function by sending an HTTP/1.1 or HTTPS POST, GET, or DELETE message to the Application Policy Infrastructure Controller (APIC). The HTML body of the POST message contains a Javascript Object Notation (JSON) or XML data structure that describes an MO or an API method. The HTML

body of the response message contains a JSON or XML structure that contains requested data, confirmation of a requested action, or error information.


Note

The root element of the response structure is imdata. This element is merely a container for the response; it is not a class in the management information model (MIM).

Configuring the HTTP Request Method and Content Type

API commands and queries must use the supported HTTP or HTTPS request methods and header fields, as described in the following sections.


Note

For security, only HTTPS is enabled as the default mode for API communications. HTTP and HTTP-to-HTTPS redirection can be enabled if desired, but are less secure. For simplicity, this document refers to HTTP in descriptions of protocol components and interactions.

Request Methods

The API supports HTTP POST, GET, and DELETE request methods as follows:

- An API command to create or update an MO, or to execute a method, is sent as an HTTP POST message.
- An API query to read the properties and status of an MO, or to discover objects, is sent as an HTTP GET message.
- An API command to delete an MO is sent as either an HTTP POST or DELETE message. In most cases, you can delete an MO by setting its status to deleted in a POST operation.

Other HTTP methods, such as PUT, are not supported.


Note

Although the DELETE method is supported, the HTTP header might show only these:
Access-Control-Allow-Methods: POST, GET, OPTIONS

Content Type

The API supports either JSON or XML data structures in the HTML body of an API request or response. You must specify the content type by terminating the URI pathname with a suffix of either .json or .xml to indicate the format to be used. The HTTP **Content-Type** and **Accept** headers are ignored by the API.

Configuring HTTP and HTTPS Using the GUI

This procedure configures the supported communication protocol for access to the GUI and the REST API.

By default, only HTTPS is enabled. HTTP or HTTP-to-HTTPS redirection, if desired, must be explicitly enabled and configured. HTTP and HTTPS can coexist.

Procedure

- Step 1** On the menu bar, click **FABRIC > Fabric Policies**.
 - Step 2** In the **Navigation** pane, expand **Pod Policies > Policies > Communication**.
 - Step 3** Under **Communication**, click the default policy.
 - Step 4** In the **Work** pane, in the HTTP or HTTPS areas, enable or disable the protocol by selecting the desired state from the **Admin State** drop-down list.
 - Step 5** In the HTTP area, enable or disable HTTP-to-HTTPS redirection by selecting the desired state from the **Redirect** drop-down list.
 - Step 6** Click **Submit**.
-

Configuring a Custom Certificate for Cisco ACI HTTPS Access Using the GUI

CAUTION: PERFORM THIS TASK ONLY DURING A MAINTENANCE WINDOW AS THERE IS A POTENTIAL FOR DOWNTIME. The downtime affects access to the APIC cluster and switches from external users or systems and not the APIC to switch connectivity. The NGINX process on the switches will also be impacted but that will be only for external connectivity and not for the fabric data plane. Access to the APIC, configuration, management, troubleshooting and such will be impacted. Expect a restart of all web servers in the fabric during this operation.

Before You Begin

Determine from which authority you will obtain the trusted certification so that you can create the appropriate Certificate Authority.

Procedure

- Step 1** On the menu bar, choose **Admin > AAA**.
- Step 2** In the **Navigation** pane, choose **Public Key Management > Certificate Authorities**.
- Step 3** In the **Work** pane, choose **Actions > Create Certificate Authority**.
- Step 4** In the **Create Certificate Authority** dialog box, in the **Name** field, enter a name for the certificate authority.
- Step 5** In the **Certificate Chain** field, copy the intermediate and root certificates for the certificate authority that will sign the Certificate Signing Request (CSR) for the Application Policy Infrastructure Controller (APIC). The certificate should be in Base64 encoded X.509 (CER) format. The intermediate certificate is placed before the root CA certificate. It should look similar to the following example:

```
-----BEGIN CERTIFICATE-----
<Intermediate Certificate>
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
<Root CA Certificate>
-----END CERTIFICATE-----
```

- Step 6** Click Submit.
- Step 7** In the **Navigation** pane, choose **Public Key Management > Key Rings**.
- Step 8** In the **Work** pane, choose **Actions > Create Key Ring**.
- Step 9** In the **Create Key Ring** dialog box, in the **Name** field, enter a name.
- Step 10** In the **Certificate** field, do not add any content.
- Step 11** In the **Modulus** field, click the radio button for the desired key strength.
- Step 12** In the **Certificate Authority** field, from the drop-down list, choose the certificate authority that you created earlier. Click **Submit**.
- Note** Do not delete the key ring. Deleting the key ring will automatically delete the associated private key used with CSRs.
- In the **Work** pane, in the **Key Rings** area, the **Admin State** for the key ring created displays **Started**.
- Step 13** In the **Navigation** pane, choose **Public Key Management > Key Rings > *key_ring_name***.
- Step 14** In the **Work** pane, choose **Actions > Create Certificate Request**.
- Step 15** In the **Subject** field, enter the fully qualified domain name (FQDN) of the APIC.
- Step 16** Fill in the remaining fields as appropriate.
- Note** Check the online help information available in the **Create Certificate Request** dialog box for a description of the available parameters.
- Step 17** Click **Submit**.
- The object is created and displayed in the **Navigation** pane under the key ring you created earlier. In the **Navigation** pane, click the object and in the **Work** pane, in the **Properties** area, in the **Request** field the CSR is displayed. Copy the contents from the field to submit to the **Certificate Authority**.
- Step 18** In the **Navigation** pane, choose **Public Key Management > Key Rings > *key_ring_name***.
- Step 19** In the **Work** pane, in the **Certificate** field, paste the signed certificate that you received from the certificate authority.
- Step 20** Click **Submit**.
- Note** If the CSR was not signed by the Certificate Authority indicated in the key ring, or if the certificate has MS-DOS line endings, an error message is displayed and the certificate is not accepted. Remove the MS-DOS line endings.
- The key is verified, and in the **Work** pane, the **Admin State** changes to **Completed** and is now ready for use in the HTTP policy.
- Step 21** On the menu bar, choose **Fabric > Fabric Policies**.
- Step 22** In the Navigation pane, choose **Pod Policies > Policies > Management Access > default**.
- Step 23** In the **Work** pane, in the **Admin Key Ring** drop-down list, choose the desired key ring.
- Step 24** Click **Submit**.
- All web servers restart. The certificate is activated, and the non-default key ring is associated with HTTPS access.

What to Do Next

You must remain aware of the expiration date of the certificate and take action before it expires. To preserve the same key pair for the renewed certificate, you must preserve the CSR as it contains the public key that pairs with the private key in the key ring. Before the certificate expires, the same CSR must be resubmitted. Do not delete or create a new key ring as deleting the key ring will delete the private key stored internally on the APIC.

Authenticating and Maintaining an API Session

Before you can access the API, you must first log in with the name and password of a configured user.

When a login message is accepted, the API returns a data structure that includes a session timeout period in seconds and a token that represents the session. The token is also returned as a cookie in the HTTP response header. To maintain your session, you must send login refresh messages to the API if no other messages are sent for a period longer than the session timeout period. The token changes each time that the session is refreshed.



Note

The default session timeout period is 300 seconds or 5 minutes.

These API methods enable you to manage session authentication:

- **aaaLogin**—Sent as a POST message, this method logs in a user and opens a session. The message body contains an aaa:User object with the name and password attributes, and the response contains a session token and cookie. If multiple AAA login domains are configured, you must prepend the user's name with **apic:domain**.
- **aaaRefresh**—Sent as a GET message with no message body or as a POST message with the **aaaLogin** message body, this method resets the session timer. The response contains a new session token and cookie.
- **aaaLogout**—Sent as a POST message, this method logs out the user and closes the session. The message body contains an aaa:User object with the name attribute. The response contains an empty data structure.
- **aaaListDomains**—Sent as a GET message, this method returns a list of valid AAA login domains. You can send this message without logging in.

You can call the authentication methods using this syntax, specifying either JSON or XML data structures:

{**http| https://host[:port]/api/methodName.{json| xml}**}

This example shows a user login message that uses a JSON data structure:

```
POST https://apic-ip-address/api/aaaLogin.json
{
    "aaaUser" : {
        "attributes" : {
            "name" : "georgewa",
            "pwd" : "password1"
        }
    }
}
```

This example shows part of the response upon a successful login, including the token and the refresh timeout period:

```
RESPONSE:
{
    "imdata" : [ {
        "aaaLogin" : {
            "attributes" : {
                "token" :
                    "GkZ15NLRZJ15+jqChouaZ9CYjgE58W/pMccR+LeXmdO0obG9NB
                    Iwo1VBo7+YC1oiJL9mS6I9qh62BkX+Xddhe0JYrTmSG4JcKZ4t3
    }]
```

```

        bcp2Mxy3VBmgoJjwZ76Zouf9V9AD6X183lyoR4bLBzqbSSU1R2N
        IgUotCGWjZt5JX6CJF0=",
        "refreshTimeoutSeconds" : "300",
        "lastName" : "Washington",
        "firstName" : "George"
    },
    "children" : [
        ...
    [TRUNCATED]
    ...
}

```

In the preceding example, the **refreshTimeoutSeconds** attribute indicates that the session timeout period is 300 seconds.

This example shows how to request a list of valid login domains:

```

GET https://apic-ip-address/api/aaaListDomains.json

RESPONSE:
{
    "imdata": [
        {
            "name": "ExampleRadius"
        },
        {
            "name": "local",
            "guiBanner": "San Jose Fabric"
        }
    ]
}

```

In the preceding example, the response data shows two possible login domains, 'ExampleRadius' and 'local.' The following example shows a user login message for the ExampleRadius login domain:

```

POST https://apic-ip-address/api/aaaLogin.json

{
    "aaaUser" : {
        "attributes" : {
            "name" : "apic:ExampleRadius\\georgewa",
            "pwd" : "paSSword1"
        }
    }
}

```

Requiring a Challenge Token for an API Session

For stronger API session security, you can require your session to use a challenge token. When you request this feature during login, the API returns a token string that you must include in every subsequent message to the API. Unlike the normal session token, the challenge token is not stored as a cookie to be automatically provided by your browser. Your API commands and queries must provide the challenge token using one of the following methods:

- The challenge token is sent as a 'challenge' parameter in the URI of your API message.
- The challenge token is part of the HTTP or HTTPS header using 'APIC-challenge'.

To initiate a session that requires a challenge token, include the URI parameter statement ?gui-token-request=yes in your login message, as shown in this example:

```
POST https://192.0.20.123/api/aaaLogin.json?gui-token-request=yes
```

The response message body contains an attribute of the form "urlToken": "*token*", where *token* is a long string of characters representing the challenge token. All subsequent messages to the API during this session must include the challenge token, as shown in this example where it is sent as a 'challenge' URI parameter:

```
GET https://192.0.20.123/api/class/aaaUser.json?challenge=fa47e44df54562c24fef6601dc...
```

This example shows how the challenge token is sent as an 'APIC-challenge' field in the HTTP header:

```
GET //api/class/aaaUser.json
HTTP/1.1
Host: 192.0.20.123
Connection: keep-alive
Accept: text/html,application/xhtml+xml,application/xml,application/json
APIC-challenge: fa47e44df54562c24fef6601dcff72259299a077336aecfc5b012b036797ab0f
.
.
```

Logging In

You can log in to the APIC REST API by sending a valid username and password in a data structure to the **aaaLogin** API method, as described in [Authenticating and Maintaining an API Session, on page 34](#). Following a successful login, you must periodically refresh the session.

The following examples show how to log in as an administrator, refresh the session during configuration, and log out using XML and JSON.



Note

At this time, the **aaaLogout** method returns a response but does not end a session. Your session ends after a refresh timeout when you stop sending **aaaRefresh** messages.

Changing Your Own User Credentials

When logged in to APIC, you can change your own user credentials, including your password, SSH key, and X.509 certificate. The following API methods are provided for changing the user credentials of the logged-in user:

- `changeSelfPassword`
- `changeSelfSshKey`
- `changeSelfX509Cert`



Note

Using these methods, you can change the credentials only for the account under which you are logged in.

The message body of each method contains the properties of the object to be modified. The properties are shown in the *Cisco APIC Management Information Model Reference*.

Changing Your Password

To change your password, send the `changeSelfPermission` API method, which modifies the `aaa:changePassword` object. The following object properties are required in the message body:

- `userName` — Your login ID.
- `oldPassword` — Your current password.
- `newPassword` — Your new password.

This example, when sent by User1, changes the password for User1.

```
POST http://192.0.20.123/api/changeSelfPermission.json

{
  "aaaChangePassword" : {
    "attributes" : {
      "userName" : "User1",
      "oldPassword" : "p@$sw0rd",
      "newPassword" : "dr0ws$@p"
    }
  }
}
```

A successful operation returns an empty `imdata` element, as in this example:

```
{
  "totalCount" : "0",
  "imdata" : []
}
```

Changing Your SSH Key

To change your SSH key, send the `changeSelfPermission` API method, which modifies the `aaa:changeSshKey` object. The following object properties are required in the message body:

- `userName` — Your login ID.
- `name` — The symbolic name of the key. APIC supports up to 32 SSH keys for a single user.
- `data` — Your new SSH key.

This example, when sent by User1, changes the SSH key for User1.

```
POST http://192.0.20.123/api/changeSshKey.json

{
  "aaaChangeSshKey" : {
    "attributes" : {
      "userName" : "User1",
      "name" : "A",
      "data" : "ssh-rsa AAAAB3NzaC1yc2EAAAQABiWAAAQEAuKxY5E4we6uCR2z== key@example.com"
    }
  }
}
```

A successful operation returns an empty `imdata` element.

Changing Your X.509 Certificate

To change your X.509 certificate, send the changeSelfX509Cert API method, which modifies the aaa:changeX509Cert object. The following object properties are required in the message body:

- `userName` — Your login ID.
- `name` — The symbolic name of the certificate. APIC supports up to 32 X.509 certificates for a single user.
- `data` — The entire data body of your new X.509 certificate.

This example, when sent by User1, changes the X.509 certificate for User1.

```
POST http://192.0.20.123/api/changeSelfX509Cert.json
{
    "aaaChangeX509Cert" : {
        "attributes" : {
            "userName" : "User1",
            "name" : "A",
            "data" : "-----BEGIN CERTIFICATE-----\nMIIE2TCCA8GgAwIBAgIKamlnsw
[EXAMPLE TRUNCATED]
                1BCIolblPFft6QKoSJFjB6thJksaE5/k3Npf\n-----END CERTIFICATE-----"
        }
    }
}
```

A successful operation returns an empty `imdata` element.

Deleting an SSH Key or X.509 Certificate

To delete a key or certificate, send the key or certificate change method with the name of the key or certificate to be deleted and with the data attribute blank.

This example, when sent by User1, deletes the SSH key for User1.

```
POST http://192.0.20.123/api/changeSelfSshKey.json
{
    "aaaChangeSshKey" : {
        "attributes" : {
            "userName" : "User1",
            "name" : "A",
            "data" : ""
        }
    }
}
```

A successful operation returns an empty `imdata` element.

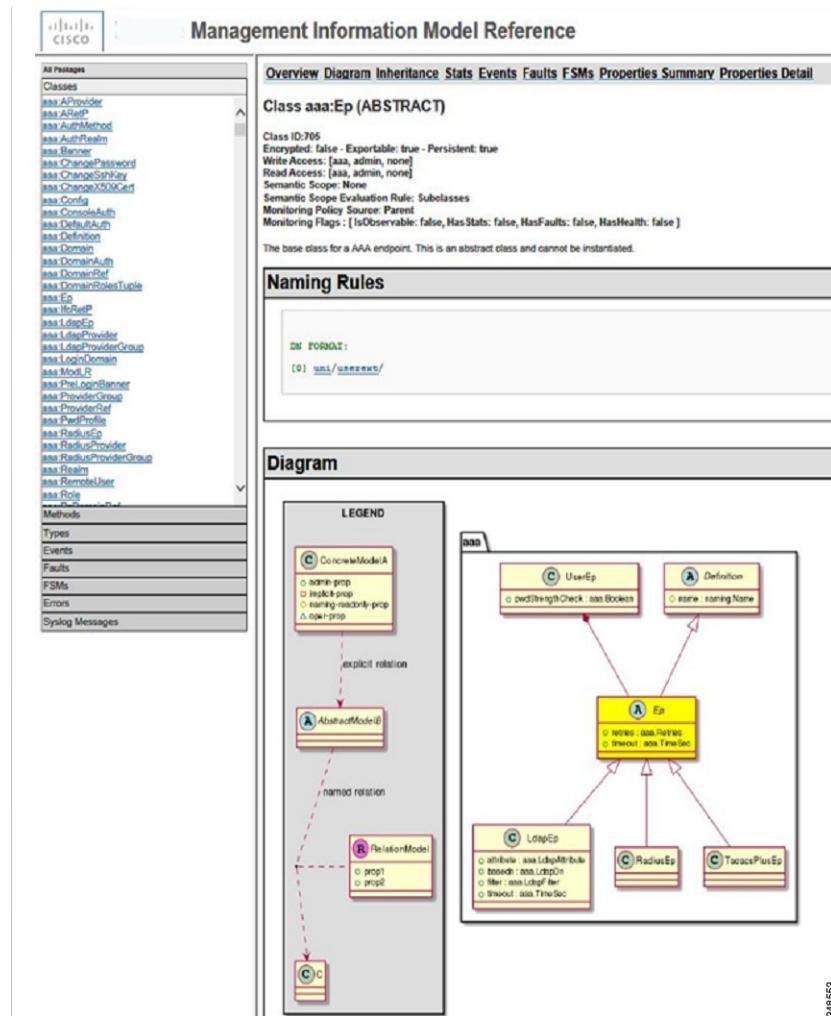
REST API Tools

Management Information Model Reference

The Management Information Model (MIM) contains all of the managed objects in the system and their properties. For details, see the *Cisco APIC Management Information Model Reference Guide*.

See the following figure for an example of how an administrator can use the MIM to research an object in the MIT.

Figure 3: MIM Reference



348553

Viewing an API Interchange in the GUI

When you perform a task in the APIC graphical user interface (GUI), the GUI creates and sends internal API messages to the operating system to execute the task. By using the API Inspector, which is a built-in tool of the APIC, you can view and copy these API messages. A network administrator can replicate these messages in order to automate key operations, or you can use the messages as examples to develop external applications that will use the API. .

Procedure

- Step 1** Log in to the APIC GUI.
- Step 2** In the upper right corner of the APIC window, click the "welcome, <name>" message to view the drop-down list.
- Step 3** In the drop-down list, choose the **Show API Inspector**.
The **API Inspector** opens in a new browser window.
- Step 4** In the **Filters** toolbar of the **API Inspector** window, choose the types of API log messages to display.
The displayed messages are color-coded according to the selected message types. This table shows the available message types:

Name	Description
trace	Displays trace messages.
debug	Displays debug messages. This type includes most API commands and responses.
info	Displays informational messages.
warn	Displays warning messages.
error	Displays error messages.
fatal	Displays fatal messages.
all	Checking this checkbox causes all other checkboxes to become checked. Unchecking any other checkbox causes this checkbox to be unchecked.

- Step 5** In the **Search** toolbar, you can search the displayed messages for an exact string or by a regular expression.
This table shows the search controls:

Name	Description
Search	In this text box, enter a string for a direct search or enter a regular expression for a regex search. As you type, the first matched field in the log list is highlighted.
Reset	Click this button to clear the contents of the Search text box.
Regex	Check this checkbox to use the contents of the Search text box as a regular expression for a search.
Match case	Check this checkbox to make the search case sensitive.
Disable	Check this checkbox to disable the search and clear the highlighting of search matches in the log list.
Next	Click this button to cause the log list to scroll to the next matched entry. This button appears only when a search is active.
Previous	Click this button to cause the log list to scroll to the previous matched entry. This button appears only when a search is active.

Name	Description
Filter	Check this checkbox to hide nonmatched lines. This checkbox appears only when a search is active.
Highlight all	Check this checkbox to highlight all matched fields. This checkbox appears only when a search is active.

Step 6 In the **Options** toolbar, you can arrange the displayed messages.

This table shows the available options:

Name	Description
Log	Check this checkbox to enable logging.
Wrap	Check this checkbox to enable wrapping of lines to avoid horizontal scrolling of the log list
Newest at the top	Check this checkbox to display log entries in reverse chronological order.
Scroll to latest	Check this checkbox to scroll immediately to the latest log entry.
Clear	Click this button to clear the log list.
Close	Click this button to close the API Inspector.

Example

This example shows two debug messages in the API Inspector window:

```
13:13:36 DEBUG - method: GET url: http://192.0.20.123/api/class/infraInfra.json
response: {"imdata": [{"infraInfra": {"attributes": {"instanceId": "0:0", "childAction": "", "dn": "uni/infra", "lcOwn": "local", "name": "", "replTs": "never", "status": ""}}}]}

13:13:40 DEBUG - method: GET url: http://192.0.20.123/api/class/l3extDomP.json?
query-target=subtree&subscription=yes
response: {"subscriptionId": "72057598349672459", "imdata": []}
```

Testing the API Using Browser Add-Ons

Using a Browser

To test an API request, you can assemble an HTTP message, send it, and inspect the response using a browser add-on utility. RESTful API clients, which are available as add-ons for most popular browsers, provide a user-friendly interface for interacting with the API. Clients include the following:

- For Firefox/Mozilla—Poster, RESTClient
- For Chrome—Advanced REST client, Postman

Browser add-ons pass the session token as a cookie so that there is no need to include the token in the payload data structure.

Testing the API with cURL

You can send API messages from a console or a command-line script using cURL, which is a tool for transferring files using URL syntax.

To send a POST message, create a file that contains the JSON or XML command body, and then enter the cURL command in this format:

```
curl -X POST --data "@<filename>" <URI>
```

You must specify the name of your descriptor file and the URI of the API operation.



Note

Make sure to include the "@" symbol before the descriptor filename.

This example creates a new tenant named ExampleCorp using the JSON data structure in the file "newtenant.json":

```
curl -X POST --data "@newtenant.json" https://apic-ip-address/api/mo/uni/tn-ExampleCorp.json
```

To send a GET message, enter the cURL command in this format:

```
curl -X GET <URI>
```

This example reads information about a tenant in JSON format:

```
curl -X GET https://apic-ip-address/api/mo/uni/tn-ExampleCorp.json
```



Note

When testing with cURL, you must log in to the API, store the authentication token, and include the token in subsequent API operations.

Related Topics

[Example: Using the JSON API to Add a User with cURL](#)

Cisco APIC Python SDK

The Python API provides a Python programming interface to the underlying REST API, allowing you to develop your own applications to control the APIC and the network fabric, enabling greater flexibility in infrastructure automation, management, monitoring and programmability.

The Python API supports Python version 2.7.

For more information, see *Cisco APIC Python SDK Documentation*, *Installing the Cisco APIC Python SDK* and <http://www.python-requests.org>.

Using the Managed Object Browser (Visore)

The Managed Object Browser, or Visore, is a utility built into the APIC that provides a graphical view of the managed objects (MOs) using a browser. The Visore utility uses the APIC REST API query methods to browse MOs active in the Application Centric Infrastructure fabric, allowing you to see the query that was used to obtain the information. The Visore utility cannot be used to perform configuration operations.


Note

Only the Firefox, Chrome, and Safari browsers are supported for Visore access.

Visore Browser Page

Filter Area

The filter form is case sensitive. This area supports all simple APIC REST API query operations.

Name	Description
Class or DN field	Object class name or fully distinguished name of a managed object.
Property field	The property of the managed object on which you want to filter the results. If you leave the Property field empty, the search returns all instances of the specific class.
Op drop-down list	Operator for the values of the property on which you want to filter the results. The following are valid operators: <ul style="list-style-type: none"> • == (equal to) • != (not equal to) • < (less than) • > (greater than) • ≤ (less than or equal to) • ≥ (greater than or equal to) • between • wildcard • anybit • allbits
Val1 field	The first value for the property on which you want to filter.
Val2 field	The second value on which you want to filter.

Display XML of Last Query Link

The **Display XML of last query** link displays the full APIC REST API translation of the most recent query run in Visore.

Results Area

You can bookmark any query results page in your browser to view the results again because the query is encoded in the URL.



Note Many of the managed objects are only used internally and are not generally applicable to APIC REST API program development.

Name	Description
Pink background	Separates individual managed object instances and displays the class name of the object below it.
Blue or green background	Indicates the property names of the managed object.
Yellow or beige background	Indicates the value of a property name.
dn property	Absolute address of each managed object in the object model.
dn link	When clicked, displays all managed objects with that dn.
Class name link	When clicked, displays all managed objects of that class.
Left arrow	When clicked, takes you to the parent object of the managed object.
Right arrow	When clicked, takes you to the child objects of the managed object.
Question mark	Links you to the XML API documentation for the managed object.

Accessing Visore

Procedure

Step 1 Open a supported browser and enter the URL of the APIC followed by `/visore.html`.

Example:

`https://apic-ip-address/visore.html`

Step 2 When prompted, log in using the same credentials you would use to log in to the APIC CLI or GUI user interfaces.

You can use a read-only account.

Running a Query in Visore

Procedure

-
- Step 1** Enter a class or DN name of the MO in the **Class or DN** text box.
- Step 2** (Optional) You can filter the query by entering a property of the MO in the **Property** text box, an operator in the **Op** text box, and one or two values in the **Val1** and **Val2** text boxes.
- Step 3** Click **Run Query**.
Visore sends a query to the APIC and the requested MO is displayed in a tabular format.
- Step 4** (Optional) Click the **Display URI of last query** link to display the API call that executed the query.
- Step 5** (Optional) Click the **Display last response** link to display the API response data structure from the query.
- Step 6** (Optional) In the **dn** field of the MO description table, click the < and > icons to retrieve the parent and child classes of the displayed MO.
Clicking > sends a query to the APIC for the children of the MO. Clicking < sends a query for the parent of the MO.
- Step 7** (Optional) In the **dn** field of the MO description table, click the additional icons to display statistics, faults, or health information for the MO.
-



PART



Part 2: Common APIC Tasks Using the REST API

- [Managing APIC Using the REST API, page 49](#)
- [Managing Roles, Users, and Signature-Based Transactions, page 71](#)
- [Common Tenant Tasks, page 81](#)
- [Managing Layer 2 Networking, page 85](#)
- [Managing Layer 3 Networking, page 91](#)
- [Monitoring Using the REST API, page 93](#)
- [Troubleshooting Using the REST API, page 99](#)



CHAPTER 2

Managing APIC Using the REST API

- [Adding Management Access, page 49](#)
- [Managing Configuration Files, page 58](#)
- [Snapshots and Rollbacks, page 64](#)
- [Using Configuration Zones, page 66](#)

Adding Management Access

In-Band and Out-of-Band Management Access

The mgmt tenant provides a convenient means to configure access to fabric management functions. While fabric management functions are accessible through the APIC, they can also be accessed directly through in-band and out-of-band network policies.

About Static Management Access

Configuring static in-band and out-of-band management connectivity is simpler than configuring dynamic in-band and out-of-band management connectivity. When configuring in-band static management, you must specify the IP address for each node and make sure to assign unique IP addresses. For simple deployments where users manage the IP addresses of a few leaf and spine switches, it is easy to configure a static management access. For more complex deployments, where you might have a large number of leaf and spine switches that require managing many IP addresses, static management access is not recommended. It is recommended that you configure a dynamic management access that automatically avoids the possible duplication of IP addresses.

**Note**

- We recommend that you configure either in-band or out-of-band static management or in-band and out-of-band dynamic management. Do not combine the two methods in your deployments.
- IPv4 and IPv6 addresses are supported for in-band management access. IPv6 configurations are supported using static configurations (for both in-band and out-of-band). IPv4 and IPv6 dual in-band and out-of-band configurations are supported only through static configuration. For more information, see the KB article, *Configuring Static Management Access in Cisco APIC*.
- Using log directive on filters in management contracts is not supported. Setting the log directive will cause zoning-rule deployment failure.

Configuring In-Band Management Access Using the REST API

IPv4 and IPv6 addresses are supported for in-band management access. IPv6 configurations are supported using static configurations (for both in-band and out-of-band). IPv4 and IPv6 dual in-band and out-of-band configurations are supported only through static configuration. For more information, see the KB article, *Configuring Static Management Access in Cisco APIC*.

Procedure

Step 1 Create a VLAN namespace.

Example:

```
POST
https://apic-ip-address/api/mo/uni.xml

<?xml version="1.0" encoding="UTF-8"?>
<!!-- api/policymgr/mo/uni.xml -->
<polUni>
  <infraInfra>
    <!-- Static VLAN range -->
    <fvnsVlanInstP name="inband" allocMode="static">
      <fvnsEncapBlk name="encap" from="vlan-10" to="vlan-11"/>
    </fvnsVlanInstP>
  </infraInfra>
</polUni>
```

Step 2 Create a physical domain.

Example:

```
POST
https://apic-ip-address/api/mo/uni.xml

<?xml version="1.0" encoding="UTF-8"?>
<!!-- api/policymgr/mo/uni.xml -->
<polUni>
  <physDomP name="inband">
    <infraRsVlanNs tDn="uni/infra/vlanns-inband-static"/>
  </physDomP>
</polUni>
```

Step 3 Create selectors for the in-band management.

Example:

```

POST
https://apic-ip-address/api/mo/uni.xml

<?xml version="1.0" encoding="UTF-8"?>
<!-- api/policymgr/mo/.xml -->
<polUni>
    <infraInfra>
        <infraNodeP name="vmmNodes">
            <infraLeafS name="leafS" type="range">
                <infraNodeBlk name="single0" from_="101" to_="101"/>
            </infraLeafS>
            <infraRsAccPortP tDn="uni/infra/accportprof-vmmPorts"/>
        </infraNodeP>

        <!-- Assumption is that VMM host is reachable via eth1/40. -->
        <infraAccPortP name="vmmPorts">
            <infraHPortS name="portS" type="range">
                <infraPortBlk name="block1"
                    fromCard="1" toCard="1"
                    fromPort="40" toPort="40"/>
            <infraRsAccBaseGrp tDn="uni/infra/funcprof/accportgrp-inband" />
        </infraHPortS>
    </infraAccPortP>

    <infraNodeP name="apicConnectedNodes">
        <infraLeafS name="leafS" type="range">
            <infraNodeBlk name="single0" from_="101" to_="102"/>
        </infraLeafS>
        <infraRsAccPortP tDn="uni/infra/accportprof-apicConnectedPorts"/>
    </infraNodeP>

    <!-- Assumption is that APIC is connected to eth1/1. -->
    <infraAccPortP name="apicConnectedPorts">
        <infraHPortS name="portS" type="range">
            <infraPortBlk name="block1"
                fromCard="1" toCard="1"
                fromPort="1" toPort="3"/>
        <infraRsAccBaseGrp tDn="uni/infra/funcprof/accportgrp-inband" />
    </infraHPortS>
    </infraAccPortP>

    <infraFuncP>
        <infraAccPortGrp name="inband">
            <infraRsAttEntP tDn="uni/infra/attentp-inband"/>
        </infraAccPortGrp>
    </infraFuncP>

    <infraAttEntityP name="inband">
        <infraRsDomP tDn="uni/phys-inband"/>
    </infraAttEntityP>
</infraInfra>
</polUni>

```

Step 4 Configure an in-band bridge domain and endpoint group (EPG).

Example:

```
POST https://apic-ip-address/api/mo/uni.xml
```

```

<?xml version="1.0" encoding="UTF-8"?>
<!-- api/policymgr/mo/.xml -->
<polUni>
    <fvTenant name="mgmt">
        <!-- Configure the in-band management gateway address on the
             in-band BD. -->
        <fvBD name="inb">
            <fvSubnet ip="10.13.1.254/24"/>
        </fvBD>
    </fvTenant>

```

```

<mgmtMgmtP name="default">
    <!-- Configure the encapsulation on which APICs will communicate on the
        in-band network. -->
    <mgmtInB name="default" encaps="vlan-10">
        <fvRsProv tnVzBrCPName="default"/>
    </mgmtInB>
</mgmtMgmtP>
</fvTenant>
</polUni>

```

Step 5 Create an address pool.

Example:

```

POST
https://apic-ip-address/api/mo/uni.xml

<?xml version="1.0" encoding="UTF-8"?>
<!-- api/policymgr/mo/.xml -->
<polUni>
    <fvTenant name="mgmt">
        <!-- Addresses for APIC in-band management network -->
        <fvnsAddrInst name="apicInb" addr="10.13.1.254/24">
            <fvnsUcastAddrBlk from="10.13.1.1" to="10.13.1.10"/>
        </fvnsAddrInst>

        <!-- Addresses for switch in-band management network -->
        <fvnsAddrInst name="switchInb" addr="10.13.1.254/24">
            <fvnsUcastAddrBlk from="10.13.1.101" to="10.13.1.120"/>
        </fvnsAddrInst>
    </fvTenant>
</polUni>

```

Note Dynamic address pools for IPv6 is not supported.

Step 6 Create management groups.

Example:

```

POST
https://apic-ip-address/api/mo/uni.xml

<?xml version="1.0" encoding="UTF-8"?>
<!-- api/policymgr/mo/.xml -->
<polUni>
    <infraInfra>
        <!-- Management node group for APICs -->
        <mgmtNodeGrp name="apic">
            <infraNodeBlk name="all" from_="1" to_="3"/>
            <mgmtRsGrp tDn="uni/infra/funcprof/grp-apic"/>
        </mgmtNodeGrp>

        <!-- Management node group for switches-->
        <mgmtNodeGrp name="switch">
            <infraNodeBlk name="all" from_="101" to_="104"/>
            <mgmtRsGrp tDn="uni/infra/funcprof/grp-switch"/>
        </mgmtNodeGrp>

        <!-- Functional profile -->
        <infraFuncP>
            <!-- Management group for APICs -->
            <mgmtGrp name="apic">
                <!-- In-band management zone -->
                <mgmtInBZone name="default">
                    <mgmtRsInbEpg tDn="uni/tn-mgmt/mgmtp-default/inb-default"/>
                    <mgmtRsAddrInst tDn="uni/tn-mgmt/addrinst-apicInb"/>
                </mgmtInBZone>
            </mgmtGrp>

            <!-- Management group for switches -->
            <mgmtGrp name="switch">

```

```
<!-- In-band management zone -->
<mgmtInBZone name="default">
    <mgmtRsInbEpg tDn="uni/tn-mgmt/mgmtp-default/inb-default"/>
    <mgmtRsAddrInst tDn="uni/tn-mgmt/addrinst-switchInb"/>
</mgmtInBZone>
</mgmtGrp>
</infraFuncP>
</infraInfra>
</polUni>
```

Note Dynamic address pools for IPv6 is not supported.

Configuring Static In-Band Management Access Using the REST API

Procedure

Step 1 Create a VLAN namespace.

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- api/policymgr/mo/uni.xml -->
<polUni>
    <infraInfra>
        <!-- Static VLAN range -->
        <fvnsVlanInstP name="inband" allocMode="static">
            <fvnsEncapBlk name="encap" from="vlan-10" to="vlan-11"/>
        </fvnsVlanInstP>
    </infraInfra>
</polUni>
```

Step 2 Create a physical domain.

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- api/policymgr/mo/uni.xml -->
<polUni>
    <physDomP name="inband">
        <infraRsVlanNs tDn="uni/infra/vlanns-inband-static"/>
    </physDomP>
</polUni>
```

Step 3 Create selectors for the in-band management.

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- api/policymgr/mo/.xml -->
<polUni>
    <infraInfra>
        <infraNodeP name="vmmNodes">
            <infraLeafS name="leafS" type="range">
                <infraNodeBlk name="single0" from_="101" to_="101"/>
            </infraLeafS>
            <infraRsAccPortP tDn="uni/infra/accportprof-vmmPorts"/>
        </infraNodeP>

        <!-- Assumption is that VMM host is reachable via eth1/40. -->
        <infraAccPortP name="vmmPorts">
            <infraHPortS name="ports" type="range">
                <infraPortBlk name="block1"
                    fromCard="1" toCard="1"
```

```

        fromPort="40" toPort="40"/>
    <infraRsAccBaseGrp tDn="uni/infra/funcprof/accportgrp-inband" />
</infraHPorts>
</infraAccPortP>

<infraNodeP name="apicConnectedNodes">
    <infraLeafS name="leafS" type="range">
        <infraNodeBlk name="single0" from_="101" to_="102"/>
    </infraLeafS>
    <infraRsAccPortP tDn="uni/infra/accportprof-apicConnectedPorts"/>
</infraNodeP>

<!-- Assumption is that APIC is connected to eth1/1. -->
<infraAccPortP name="apicConnectedPorts">
    <infraHPorts name="portsS" type="range">
        <infraPortBlk name="block1"
            fromCard="1" toCard="1"
            fromPort="1" toPort="3"/>
        <infraRsAccBaseGrp tDn="uni/infra/funcprof/accportgrp-inband" />
    </infraHPorts>
</infraAccPortP>

<infraFuncP>
    <infraAccPortGrp name="inband">
        <infraRsAttEntP tDn="uni/infra/attentp-inband"/>
    </infraAccPortGrp>
</infraFuncP>

<infraAttEntityP name="inband">
    <infraRsDomP tDn="uni/phys-inband"/>
</infraAttEntityP>
</infraInfra>
</polUni>

```

Step 4 Configure an in-band bridge domain and endpoint group (EPG).

Example:

```

<?xml version="1.0" encoding="UTF-8"?>
<!-- api/policymgr/mo/.xml -->
<polUni>
    <fvTenant name="mgmt">
        <!-- Configure the in-band management gateway address on the
             in-band BD. -->
        <fvBD name="inb">
            <fvSubnet ip="subnet_ip_address" />
        </fvBD>

        <mgmtMgmtP name="default">
            <!-- Configure the encap on which APICs will communicate on the
                 in-band network. -->
            <mgmtInB name="default" encap="vlan-10">
                <fvRsProv tnVzBrCPName="default"/>
            </mgmtInB>
        </mgmtMgmtP>
    </fvTenant>
</polUni>

```

Step 5 Create static in-band management IP addresses and assign them to node IDs.

Example:

```

<polUni>
    <fvTenant name="mgmt">
        <mgmtMgmtP name="default">
            <mgmtInB name="default">
                <mgmtRsInBStNode tDn="topology/pod-1/node-101"
                    addr="ip_address_1"
                    gw="gw_address">
                    v6Addr = "<ip6_address_1>"
                    v6Gw = "<ip6_gw_address>"/>
            </mgmtInB>
        </mgmtMgmtP>
    </fvTenant>
</polUni>

```

```

<mgmtRsInBStNode tDn="topology/pod-1/node-102"
    addr="<ip_address_2>"
    gw="<gw_address>">
    v6Addr = "<ip6_address_2>">
    v6Gw = "<ip6_gw_address>/>
        <mgmtRsInBStNode tDn="topology/pod-1/node-103"
            addr="<ip_address_3>">
            gw="<gw_address>">
            v6Addr = "<ip6_address_3>">
            v6Gw = "<ip6_gw_address>/>
                <mgmtRsInBStNode tDn="topology/pod-1/node-104"
                    addr="<ip_address_4>">
                    gw="<gw_address>">
                    v6Addr = "<ip6_address_4>">
                    v6Gw = "<ip6_gw_address>/>

                    <mgmtRsInBStNode tDn="topology/pod-1/node-105"
                        addr="<ip_address_5>">
                        gw="<gw_address>">
                        v6Addr = "<ip6_address_5>">
                        v6Gw = "<ip6_gw_address>/>

                    </mgmtInB>
                </mgmtMgmtP>
            </fvTenant>
        </polUni>

```

Configuring Out-of-Band Management Access Using the REST API

IPv4 and IPv6 addresses are supported for out-of-band management access.

Before You Begin

The APIC out-of-band management connection link must be 1 Gbps.

Procedure

- Step 1** Create an out-of-band contract.

Example:

POST <https://apic-ip-address/api/mo/uni.xml>

```

<polUni>
    <fvTenant name="mgmt">
        <!-- Contract -->
        <vzOOBBrCP name="oob-default">
            <vzSubj name="oob-default">
                <vzRsSubjFiltAtt tnVzFilterName="default" />
            </vzSubj>
        </vzOOBBrCP>
    </fvTenant>
</polUni>

```

- Step 2** Associate the out-of-band contract with an out-of-band EPG.

Example:

POST <https://apic-ip-address/api/mo/uni.xml>

```
<polUni>
```

```

<fvTenant name="mgmt">
    <mgmtMgmtP name="default">
        <mgmtOoB name="default">
            <mgmtRsOoBProv tnVzOOBBrCPName="oob-default" />
        </mgmtOoB>
    </mgmtMgmtP>
</fvTenant>
</polUni>

```

Step 3 Associate the out-of-band contract with an external management EPG.

Example:

POST <https://apic-ip-address/api/mo/uni.xml>

```

<polUni>
    <fvTenant name="mgmt">
        <mgmtExtMgmtEntity name="default">
            <mgmtInstP name="oob-mgmt-ext">
                <mgmtRsOoBCons tnVzOOBBrCPName="oob-default" />
                <!-- SUBNET from where switches are managed -->
                <mgmtSubnet ip="10.0.0.0/8" />
            </mgmtInstP>
        </mgmtExtMgmtEntity>
    </fvTenant>
</polUni>

```

Step 4 Create a management address pool.

Example:

POST <https://apic-ip-address/api/mo/uni.xml>

```

<polUni>
    <fvTenant name="mgmt">
        <fvnsAddrInst name="switchOobobaddr" addr="172.23.48.1/21">
            <fvnsUcastAddrBlk from="172.23.49.240" to="172.23.49.244"/>
        </fvnsAddrInst>
    </fvTenant>
</polUni>

```

Step 5 Create node management groups.

Example:

POST <https://apic-ip-address/api/mo/uni.xml>

```

<polUni>
    <infraInfra>
        <infraFuncP>
            <mgmtGrp name="switchOob">
                <mgmtOoBZone name="default">
                    <mgmtRsAddrInst tDn="uni/tn-mgmt/addrinst-switchOobobaddr" />
                    <mgmtRsOobEpg tDn="uni/tn-mgmt/mgmtp-default/oob-default" />
                </mgmtOoBZone>
            </mgmtGrp>
        </infraFuncP>
        <mgmtNodeGrp name="switchOob">
            <mgmtRsGrp tDn="uni/infra/funcprof/grp-switchOob" />
            <infraNodeBlk name="default" from_="101" to_="103" />
        </mgmtNodeGrp>
    </infraInfra>
</polUni>

```

Note You can configure the APIC server to use out-of-band management connectivity as the default connectivity mode.

```
POST https://apic-ip-address/api/node/mo/.xml
<polUni>
  <fabricInst>
    <mgmtConnectivityPrefs interfacePref="ooband"/>
  </fabricInst>
</polUni>
```

Configuring Static Out-of-Band Management Access Using the REST API

Before You Begin

The APIC out-of-band management connection link must be 1 Gbps.

Procedure

Step 1 Create an out-of-band contract.

Example:

```
<polUni>
  <fvTenant name="mgmt">
    <!-- Contract -->
    <vzOOBBrCP name="oob-default">
      <vzSubj name="oob-default">
        <vzRsSubjFiltAtt tnVzFilterName="default" />
      </vzSubj>
    </vzOOBBrCP>
  </fvTenant>
</polUni>
```

Step 2 Associate the out-of-band contract with an out-of-band EPG.

Example:

```
<polUni>
  <fvTenant name="mgmt">
    <mgmtMgmtP name="default">
      <mgmtOoB name="default">
        <mgmtRsOoBProv tnVzOOBBrCPName="oob-default" />
      </mgmtOoB>
    </mgmtMgmtP>
  </fvTenant>
</polUni>
```

Step 3 Associate the out-of-band contract with an external management EPG.

Example:

```
<polUni>
  <fvTenant name="mgmt">
    <mgmtExtMgmtEntity name="default">
      <mgmtInstP name="oob-mgmt-ext">
        <mgmtRsOoBCons tnVzOOBBrCPName="oob-default" />
        <!-- SUBNET from where switches are managed -->
        <mgmtSubnet ip="" />
      </mgmtInstP>
    </mgmtExtMgmtEntity>
```

```
</fvTenant>
</polUni>
```

- Step 4** Create static out-of-band management IP addresses and assign them to node IDs.
CHECK IP Addresses

Example:

```
<polUni>
  <fvTenant name="mgmt">
    <mgmtMgmtP name="default">
      <mgmtOoB name="default">
        <mgmtRsOoBStNode tDn="topology/pod-1/node-101"
          addr="<ip_address_1>"
          gw="<gw_address>"/>
        <mgmtRsOoBStNode tDn="topology/pod-1/node-102"
          addr="<ip_address_2>"
          gw="<gw_address>"/>
        <mgmtRsOoBStNode tDn="topology/pod-1/node-103"
          addr="<ip_address_3>"
          gw="<gw_address>"/>
      </mgmtOoB>
    </mgmtMgmtP>
  </fvTenant>
</polUni>
```

Managing Configuration Files

Overview

This topic provides information on:

- How to use configuration Import and Export to recover configuration states to the last known good state using the Cisco APIC
- How to encrypt secure properties of Cisco APIC configuration files

You can do both scheduled and on-demand backups of user configuration. Recovering configuration states (also known as "roll-back") allows you to go back to a known state that was good before. The option for that is called an Atomic Replace. The configuration import policy (configImportP) supports atomic + replace (importMode=atomic, importType=replace). When set to these values, the imported configuration overwrites the existing configuration, and any existing configuration that is not present in the imported file is deleted. As long as you do periodic configuration backups and exports, or explicitly trigger export with a known good configuration, then you can later restore back to this configuration using the following procedures for the CLI, REST API, and GUI.

For more detailed conceptual information about recovering configuration states using the Cisco APIC, please refer to the *Cisco Application Centric Infrastructure Fundamentals Guide*.

The following section provides conceptual information about encrypting secure properties of configuration files:

Backing Up, Restoring, and Rolling Back Configuration Files Workflow

This section describes the workflow of the features for backing up, restoring, and rolling back configuration files. All of the features described in this document follow the same workflow pattern. Once the corresponding policy is configured, **adminIntSt** must be set to **triggered** in order to trigger the job.

Once triggered, an object of type **configJob** (representing that run) is created under a container object of type **configJobCont**. (The naming property value is set to the policy DN.) The container's **lastJobName** field can be used to determine the last job that was triggered for that policy.



Note

Up to five **configJob** objects are kept under a single job container at a time, with each new job triggered. The oldest job is removed to ensure this.

The **configJob** object contains the following information:

- Execution time
- Name of the file being processed/generated
- Status, as follows:
 - Pending
 - Running
 - Failed
 - Fail-no-data
 - Success
 - Success-with-warnings
- Details string (failure messages and warnings)
- Progress percentage = $100 * \text{lastStepIndex}/\text{totalStepCount}$
- Field **lastStepDescr** indicating what was being done last

About Configuration Export to Controllers

Configuration export extracts user-configurable managed object (MO) trees from all 32 shards in the cluster, writes them into separate files, then compresses them into a tar gzip file. The configuration export then uploads the tar gzip file to a preconfigured remote location (configured through **configRsRemotePath** pointing to a **fileRemotePath** object) or stores it as a **snapshot** on the controller(s).



Note

See the Snapshots section for more details.

The **configExportP** policy is configured as follows:

- **name**—Policy name.
- **format**—Format in which the data is stored inside the exported archive (xml or json).

- **targetDn**—The domain name (DN) of the specific object you want to export. (Empty means everything.)
- **snapshot**—When true, the file is stored on the controller; no remote location configuration is needed.
- **includeSecureFields**—Set to true by default, this indicates whether the encrypted fields (passwords, etc.) should be included in the export archive.

**Note**

The **configSnapshot** object is created holding the information about this snapshot. (See the Snapshots section.)

Scheduling Exports

An export policy can be linked with a scheduler, which triggers the export automatically based on a preconfigured schedule. This is done through the **configRsExportScheduler** relation from the policy to a **trigSchedP** object. (See the Sample Configuration section.)

**Note**

A scheduler is optional. A policy can be triggered at any time by setting the adminSt to **triggered**.

About Configuration Import to Controller

Configuration import downloads, extracts, parses, analyzes, and applies the specified, previously exported archive one shard at a time in the following order: infra, fabric, tn-common, then everything else. The fileRemotePath configuration is performed the same way as for export (through configRsRemotePath). Importing snapshots is also supported.

The **configImportP** policy is configured as follows:

- **name**—Policy name
- **fileName**—Name of the archive file (not the path file) to be imported
- **importMode**
 - Best-effort mode: Each MO is applied individually, and errors only cause the invalid MOs to be skipped.

**Note**

If the object is not present on the controller, none of the children of the object get configured. Best-effort mode attempts to configure the children of the object.

- Atomic mode: configuration is applied by whole shards. A single error causes the whole shard to be rolled back to its original state.

- **importType**

- Replace—Current system configuration is replaced with the contents or the archive being imported. (Only atomic mode is supported.)
- Merge—Nothing is deleted, and archive content is applied on top the existing system configuration.

- **snapshot**—When true, the file is taken from the controller and no remote location configuration is needed.
- **failOnDecryptErrors**—(true by default) The file fails to import if the archive was encrypted with a different key than the one that is currently set up in the system.

Troubleshooting

The following scenarios may need troubleshooting:

- If the generated archive could not be downloaded from the remote location, refer to the Connectivity Issues section.
- If the import succeeded with warnings, check the details.
- If a file could not be parsed, refer to the following scenarios:
 - If the file is not a valid XML or JSON file, check whether the files from the exported archive were manually modified.
 - If an object property has an unknown property or property value, it may be because:
 - The property was removed or an unknown property value was manually entered.
 - The model type range was modified (non-backward compatible model change).
 - The naming property list was modified.
- If an MO could not be configured, note the following:
 - Best-effort mode logs the error and skips the MO.
 - Atomic mode logs the error and skips the shard.

Configuration File Encryption

As of release 1.1(2), the secure properties of APIC configuration files can be encrypted by enabling AES-256 encryption. AES encryption is a global configuration option; all secure properties conform to the AES configuration setting. It is not possible to export a subset of the ACI fabric configuration such as a tenant configuration with AES encryption while not encrypting the remainder of the fabric configuration. See the Cisco Application Centric Infrastructure Fundamentals Appendix K: Secure Properties for the list of secure properties.

The APIC uses a 16 to 32 character passphrase to generate the AES-256 keys. The APIC GUI displays a hash of the AES passphrase. This hash can be used to see if the same passphrases was used on two ACI fabrics. This hash can be copied to a client computer where it can be compared to the passphrase hash of another ACI fabric to see if they were generated with the same passphrase. The hash cannot be used to reconstruct the original passphrase or the AES-256 keys.

Observe the following guidelines when working with encrypted configuration files:

- Backward compatibility is supported for importing old ACI configurations into ACI fabrics that use the AES encryption configuration option.

**Note**

Reverse compatibility is not supported; configurations exported from ACI fabrics that have enabled AES encryption cannot be imported into older versions of the APIC software.

- Always enable AES encryption when performing fabric backup configuration exports. Doing so will assure that all the secure properties of the configuration will be successfully imported when restoring the fabric.

**Note**

If a fabric backup configuration is exported without AES encryption enabled, none of the secure properties will be included in the export. Since such an unencrypted backup would not include any of the secure properties, it is possible that importing such a file to restore a system could result in the administrator along with all users of the fabric being locked out of the system.

- The AES passphrase that generates the encryption keys cannot be recovered or read by an ACI administrator or any other user. The AES passphrase is not stored. The APIC uses the AES passphrase to generate the AES keys, then discards the passphrase. The AES keys are not exported. The AES keys cannot be recovered since they are not exported and cannot be retrieved via the REST API.
- The same AES-256 passphrase always generates the same AES-256 keys. Configuration export files can be imported into other ACI fabrics that use the same AES passphrase.
- For troubleshooting purposes, export a configuration file that does not contain the encrypted data of the secure properties. Temporarily turning off encryption before performing the configuration export removes the values of all secure properties from the exported configuration. To import such a configuration file that has all secure properties removed, use the import merge mode; do not use the import replace mode. Using the import merge mode will preserve the existing secure properties in the ACI fabric.
- By default, the APIC rejects configuration imports of files that contain fields that cannot be decrypted. Use caution when turning off this setting. Performing a configuration import inappropriately when this default setting is turned off could result in all the passwords of the ACI fabric to be removed upon the import of a configuration file that does not match the AES encryption settings of the fabric.

**Note**

Failure to observe this guideline could result in all users, including fabric administrations, being locked out of the system.

About the fileRemotePath Object

The fileRemotePath object holds the following remote location-path parameters:

- Hostname or IP
- Port
- Protocol: FTP, SCP, and others
- Remote directory (not file path)

- Username
- Password



Note The password must be resubmitted every time changes are made.

Sample Configuration

The following is a sample configuration:

Under **fabricInst** (uni/fabric), enter:

```
<fileRemotePath name="path-name" host="host name or ip" protocol="scp"
remotePath="path/to/some/folder" userName="user-name" userpasswd="password" />
```

Configuring a Remote Location Using the REST API

This procedure explains how to create a remote location using the REST API.

```
<fileRemotePath name="local" host="host or ip" protocol="ftp|scp|sftp" remotePath="path to
folder" userName="uname" userPasswd="pwd" />
```

Configuring Configuration File Export to Controller Using the REST API

Before You Begin

Create a remote path and scheduling policy.



Note When providing a remote location, if you set the snapshot to `True`, the backup ignores the remote path and stores the file on the controller.

Procedure

Create a configuration export policy by sending a POST request with XML such as the following example.

Example:

```
<configExportP name="policy-name" format="xml" targetDn="/some/dn or empty which means
everything"
snapshot="false" adminSt="triggered">
<configRsRemotePath tnFileRemotePathName="some remote path name" />
<configRsExportScheduler tnTrigSchedPName="some scheduler name" />
</configExportP>
```

Configuring a Configuration File Import Policy Using the REST API

Procedure

Configure a configuration file import policy, send a post with XML such as the following example:

Example:

```
<configImportP name="policy-name" fileName="someexportfile.tgz" importMode="atomic"
    importType="replace" snapshot="false" adminSt="triggered">
<configRsRemotePath tnFileRemotePathName="some remote path name" />
</configImportP>
```

Encrypting Configuration Files Using the REST API

Procedure

To encrypt a configuration file using the REST API, send a post with XML such as the following example:

Example:

```
https://apic-ip-address/api/mo/uni/fabric.xml
<pkiExportEncryptionKey passphrase="abcdefghijklmnopqrstuvwxyz"
strongEncryptionEnabled="true"/>
```

Snapshots and Rollbacks

Snapshots

Snapshots are configuration backup archives, stored (and replicated) in a controller managed folder. To create one, an export can be performed with the **snapshot** property set to true. In this case, no remote path configuration is needed. An object of **configSnapshot** type is created to expose the snapshot to the user.

configSnapshot objects provide the following:

- file name
- file size
- creation date
- root DN indicating what the snapshot is of (fabric, infra, specific tenant, and so on)
- ability to remove a snapshot (by setting the retire field to true)

To import a snapshot, set the import policy snapshot property to true and provide the name of the snapshot file (from configSnapshot).

About Rollbacks

The **configRollbackP** policy is used to undo the changes made between two snapshots. Managed Objects (MOs) are processed as follows:

- Deleted MOs are recreated.
- Created MOs are deleted.
- Modified MOs are reverted.

**Note**

The rollback feature operates only on snapshots. Remote archives are not supported. If you want to use the data in a remote archive, use the snapshot manager to create a snapshot from the data for the rollback. The policy does not require a remote path configuration.

Rollback Workflow

The policy snapshotOneDN and snapshotTwoDn fields must be set and the first snapshot (S1) must precede snapshot two (S2). Once triggered, snapshots are extracted and analyzed, and the difference between them is calculated and applied.

MOs are located that are:

- Present in S1 but not present in S2—These MOs are deleted and rollback re-creates them.
- Not present in S1 but not present in S2—These MOs are created after S1 and rollback deletes them if:
 - These MOs are not modified after S2 is taken.
 - None of the MO descendants are created or modified after S2 is taken.
- Present in both S1 and S2, but with different property values—These MO properties are reverted to S1, unless the property was modified to a different value after S2 is taken. In this case, it is left as is.

The rollback feature also generates a diff file that contains the configuration generated as a result of these calculations. Applying this configuration is the last step of the rollback process. The content of this file can be retrieved through a special REST API called readdiff:
`apichost/mqapi2/snapshots.readdiff.xml?jobdn=SNAPSHOT_JOB_DN`.

Rollback (which is difficult to predict) also has a preview mode (set preview to true), which prevents rollback from making any actual changes. It calculates and generates the diff file, allowing you to preview what exactly is going to happen once the rollback is actually performed.

Diff Tool

Another special REST API is available, which provides diff functionality between two snapshots:
`apichost/mqapi2/snapshots.diff.xml?s1dn=SNAPSHOT_ONE_DN&s2dn=SNAPSHOT_TWO_DN`.

Uploading and Downloading Snapshots Using the REST API

The **configSnapshotManagerP** policy allows you to create snapshots from remotely stored export archives. You can attach a remote path to the policy, provide the file name (same as with configImportP), set the mode to download, and trigger. The manager downloads the file, analyzes it to make sure that the archive is valid, stores it on the controller, and creates the corresponding configSnapshot object. The snapshot manager also allow you to upload a snapshot archive to a remote location. In this case, the mode must be set to upload.

Before You Begin

Set up remotely stored archives.

Procedure

To download or upload a snapshot policy, send a POST request with XML such as the following:

Example:

```
<configSnapshotManagerP name="policy-name" fileName="someexportfile.tgz"
    mode="upload|download" adminSt="triggered">
<configRsRemotePath tnFileRemotePathName="some remote path name" />
</configSnapshotManagerP>
```

Configuring and Executing a Configuration Rollback Using the REST API

Before You Begin

Create a rollback policy and a snapshot.

Procedure

To configure and execute a rollback, send a POST request with XML such as the following:

Example:

```
<configRollbackP name="policy-name" snapshotOneDn="dn/of/snapshot/one"
snapshotOneDn="dn/of/snapshot/two" preview="false" adminSt="triggered" />
```

Using Configuration Zones

Configuration Zones

Configuration zones divide the ACI fabric into different zones that can be updated with configuration changes at different times. This limits the risk of deploying a faulty fabric-wide configuration that might disrupt traffic or even bring the fabric down. An administrator can deploy a configuration to a non-critical zone, and then deploy it to critical zones when satisfied that it is suitable.

The following policies specify configuration zone actions:

- `infrazone:ZoneP` is automatically created upon system upgrade. It cannot be deleted or modified.
- `infrazone:Zone` contains one or more pod groups (`PodGrp`) or one or more node groups (`NodeGrp`).

**Note**

You can only choose `PodGrp` or `NodeGrp`; both cannot be chosen.

A node can be part of only one zone (`infrazone:Zone`). `NodeGrp` has two properties: name, and deployment mode. The deployment mode property can be:

- `enabled` - Pending updates are sent immediately.
- `disabled` - New updates are postponed.

**Note**

Do not upgrade, downgrade, commission, or decommission nodes in a disabled configuration zone.

- `triggered` - pending updates are sent immediately, and the deployment mode is automatically reset to the value it had before the change to `triggered`.

When a policy on a given set of nodes is created, modified, or deleted, updates are sent to each node where the policy is deployed. Based on policy class and `infrazone` configuration the following happens:

- For policies that do not follow `infrazone` configuration, the APIC sends updates immediately to all the fabric nodes.
- For policies that follow `infrazone` configuration, the update proceeds according to the `infrazone` configuration:
 - If a node is part of an `infrazone:Zone`, the update is sent immediately if the deployment mode of the zone is set to enabled; otherwise the update is postponed.
 - If a node is not part of an `infrazone:Zone`, the update is done immediately, which is the ACI fabric default behavior.

Configuration Zone Supported Policies

The following policies are supported for configuration zones:

```
analytics:CfgSrv
bgp:InstPol
callhome:Group
callhome:InvP
callhome:QueryGroup
cdp:IfPol
cdp:InstPol
comm:Pol
comp:DomP
coop:Pol
datetime:Pol
dbgexp:CoreP
dbgexp:TechSupP
dhcp:NodeGrp
dhcp:PodGrp
edr:ErrDisRecoverPol
ep:ControlP
ep:LoopProtectP
eqptdiagp:TsOdFabP
eqptdiagp:TsOdLeafP
fabric:AutoGEp
fabric:ExplicitGEp
fabric:FuncP
fabric:HifPol
fabric:L1IfPol
fabric:L2IfPol
fabric:L2InstPol
fabric:L2PortSecurityPol
fabric:LeCardP
fabric:LeCardPGP
fabric:LeCardS
fabric:LeNodePGP
fabric:LePortP
fabric:LePortPGP
fabric:LFPorts
fabric:NodeControl
fabric:OLeafs
fabric:OSpines
fabric:PodPGP
fabric:PortBlk
fabric:ProtGEp
fabric:ProtPol
```

```
fabric:SFPorts
fabric:SpCardP
fabric:SpCardPGrp
fabric:SpCards
fabric:SpNodePGrp
fabric:SpPortP
fabric:SpPortPGrp
fc:DomP
fc:FabricPol
fc:IfPol
fc:InstPol
file:RemotePath
fvns:McastAddrInstP
fvns:VlanInstP
fvns:VsanInstP
fvns:VxlanInstP
infra:AccBaseGrp
infra:AccBndlGrp
infra:AccBndlPolGrp
infra:AccBndlSubgrp
infra:AccCardP
infra:AccCardPGrp
infra:AccNodePGrp
infra:AccPortGrp
infra:AccPortP
infra:AttEntityP
infra:Cards
infra:ConnFexBlk
infra:ConnFexS
infra:ConnNodeS
infra:DomP
infra:FexBlk
infra:FexBndlGrp
infra:FexGrp
infra:FexP
infra:FuncP
infra:HConnPorts
infra:HPaths
infra:HPorts
infra:Leafs
infra:NodeBlk
infra:NodeGrp
infra:NodeP
infra:OLeafs
infra:OSpineS
infra:PodBlk
infra:PodGrp
infra:PodP
infra:PodS
infra:PolGrp
infra:PortBlk
infra:PortP
infra:Ports
infra:PortTrackPol
infra:Profile
infra:SHPaths
infra:SHPorts
infra:SpAccGrp
infra:SpAccPortGrp
infra:SpAccPortP
infra:SpineP
infra:SpineS
isis:DomPol
l2ext:DomP
l2:IfPol
l2:InstPol
l2:PortSecurityPol
l3ext:DomP
lacp:IfPol
lacp:LagPol
lldp:IfPol
lldp:InstPol
mcp:IfPol
```

```

mcp:InstPol
mgmt:NodeGrp
mgmt:PodGrp
mon:FabricPol
mon:InfraPol
phys:DomP
psu:InstPol
qos:DppPol
snmp:Pol
span:Dest
span:DestGrp
span:SpanProv
span:SrcGrp
span:SrcTargetShadow
span:SrcTargetShadowBD
span:SrcTargetShadowCtx
span:TaskParam
span:VDest
span:VDestGrp
span:VSpanProv
span:VSrcGrp
stormctrl:IfPol
stp:IfPol
stp:InstPol
stp:MstDomPol
stp:MstRegionPol
trig:SchedP
vmm:DomP
vpc:InstPol
vpc:KAPol

```

Creating Configuration Zones Using the REST API

Before You Begin

This procedure explains how to create a configuration zone using the REST API.

Procedure

Create a configuration zone using the REST API leaf switch or pod examples below.

Example:

Creating a Config Zone with Leaf Switches

```

<infraInfra>
<infrazoneZoneP name="default">
<infrazoneZone name="Group1" deplMode="disabled">
<infrazoneNodeGrp name="nodeGroup">
<infraNodeBlk name="nodeblk1" from_=101 to_=101/>
<infraNodeBlk name="nodeblk2" from_=103 to_=103/>
</infrazoneNodeGrp>
</infrazoneZone>
<infrazoneZone name="Group2" deplMode="enabled">
<infrazoneNodeGrp name="nodeGroup2">
<infraNodeBlk name="nodeblk" from_=102 to_=102/>
</infrazoneNodeGrp>
</infrazoneZone>
</infrazoneZoneP>
</infraInfra>

```

Example:

Creating a Config Zone with Pods

```
<infraInfra>
  <infrazoneZoneP name="default">
    <infrazoneZone name="testZone" descr="testZone-Description" deplMode="enabled">
      <infrazonePodGrp name="podGroup1">
        <infraPodBlk name="group1" from_=101 to_=101/>
        <infraPodBlk name="group2" from_=103 to_=103/>
      </infrazonePodGrp>
      <infrazonePodGrp name="podGroup2">
        <infraPodBlk name="group" from_=102 to_=102/>
      </infrazonePodGrp>
    </infrazoneZone>
  </infrazoneZoneP>
</infraInfra>
```



CHAPTER 3

Managing Roles, Users, and Signature-Based Transactions

- [Managing APIC Roles and Users, page 71](#)
- [APIC Signature-Based Transactions, page 76](#)

Managing APIC Roles and Users

User Access, Authorization, and Accounting

Application Policy Infrastructure Controller (APIC) policies manage the authentication, authorization, and accounting (AAA) functions of the Cisco Application Centric Infrastructure (ACI) fabric. The combination of user privileges, roles, and domains with access rights inheritance enables administrators to configure AAA functions at the managed object level in a granular fashion. These configurations can be implemented using the REST API, the CLI, or the GUI.

Access Rights Workflow Dependencies

The Cisco Application Centric Infrastructure (ACI) RBAC rules enable or restrict access to some or all of the fabric. For example, in order to configure a leaf switch for bare metal server access, the logged in administrator must have rights to the `infra` domain. By default, a tenant administrator does not have rights to the `infra` domain. In this case, a tenant administrator who plans to use a bare metal server connected to a leaf switch could not complete all the necessary steps to do so. The tenant administrator would have to coordinate with a fabric administrator who has rights to the `infra` domain. The fabric administrator would set up the switch configuration policies that the tenant administrator would use to deploy an application policy that uses the bare metal server attached to an ACI leaf switch.

Accounting

ACI fabric accounting is handled by these two managed objects (MO) that are processed by the same mechanism as faults and events:

- The `aaaSessionLR` MO tracks user account login and logout sessions on the APIC and switches, and token refresh. The ACI fabric session alert feature stores information such as the following:
 - Username
 - IP address initiating the session
 - Type (telnet, https, REST etc.)
 - Session time and length
 - Token refresh – a user account login event generates a valid active token which is required in order for the user account to exercise its rights in the ACI fabric.

**Note**

Token expiration is independent of login; a user could log out but the token expires according to the duration of the timer value it contains.

- The `aaaModLR` MO tracks the changes users make to objects and when the changes occurred.
- If the AAA server is not pingable, it is marked unavailable and a fault is seen.

Both the `aaaSessionLR` and `aaaModLR` event logs are stored in APIC shards. Once the data exceeds the pre-set storage allocation size, it overwrites records on a first-in first-out basis.

**Note**

In the event of a destructive event such as a disk crash or a fire that destroys an APIC cluster node, the event logs are lost; event logs are not replicated across the cluster.

The `aaaModLR` and `aaaSessionLR` MOs can be queried by class or by distinguished name (DN). A class query provides all the log records for the whole fabric. All `aaaModLR` records for the whole fabric are available from the GUI at the **Fabric > Inventory > POD > History > Audit Log** section. The APIC GUI **History > Audit Log** options enable viewing event logs for a specific object identified in the GUI.

The standard syslog, callhome, REST query, and CLI export mechanisms are fully supported for `aaaModLR` and `aaaSessionLR` MO query data. There is no default policy to export this data.

There are no pre-configured queries in the APIC that report on aggregations of data across a set of objects or for the entire system. A fabric administrator can configure export policies that periodically export `aaaModLR` and `aaaSessionLR` query data to a syslog server. Exported data can be archived periodically and used to generate custom reports from portions of the system or across the entire set of system logs.

Multiple Tenant Support

A core Application Policy Infrastructure Controller (APIC) internal data access control system provides multitenant isolation and prevents information privacy from being compromised across tenants. Read/write restrictions prevent any tenant from seeing any other tenant's configuration, statistics, faults, or event data. Unless the administrator assigns permissions to do so, tenants are restricted from reading fabric configuration, policies, statistics, faults, or events.

User Access: Roles, Privileges, and Security Domains

The APIC provides access according to a user's role through role-based access control (RBAC). An Cisco Application Centric Infrastructure (ACI) fabric user is associated with the following:

- A set of roles
- For each role, a privilege type: no access, read-only, or read-write
- One or more security domain tags that identify the portions of the management information tree (MIT) that a user can access

The ACI fabric manages access privileges at the managed object (MO) level. A privilege is an MO that enables or restricts access to a particular function within the system. For example, fabric-equipment is a privilege bit. This bit is set by the Application Policy Infrastructure Controller (APIC) on all objects that correspond to equipment in the physical fabric.

A role is a collection of privilege bits. For example, because an "admin" role is configured with privilege bits for "fabric-equipment" and "tenant-security," the "admin" role has access to all objects that correspond to equipment of the fabric and tenant security.

A security domain is a tag associated with a certain subtree in the ACI MIT object hierarchy. For example, the default tenant "common" has a domain tag `common`. Similarly, the special domain tag `all` includes the entire MIT object tree. An administrator can assign custom domain tags to the MIT object hierarchy. For example, an administrator could assign the "solar" domain tag to the tenant named solar. Within the MIT, only certain objects can be tagged as security domains. For example, a tenant can be tagged as a security domain but objects within a tenant cannot.

Creating a user and assigning a role to that user does not enable access rights. It is necessary to also assign the user to one or more security domains. By default, the ACI fabric includes two special pre-created domains:

- `All`—allows access to the entire MIT
- `Infra`— allows access to fabric infrastructure objects/subtrees, such as fabric access policies



Note

For read operations to the managed objects that a user's credentials do not allow, a "DN/Class Not Found" error is returned, not "DN/Class Unauthorized to read." For write operations to a managed object that a user's credentials do not allow, an HTTP 401 Unauthorized error is returned. In the GUI, actions that a user's credentials do not allow, either they are not presented, or they are grayed out.

A set of predefined managed object classes can be associated with domains. These classes should not have overlapping containment. Examples of classes that support domain association are as follows:

- Layer 2 and Layer 3 network managed objects
- Network profiles (such as physical, Layer 2, Layer 3, management)
- QoS policies

When an object that can be associated with a domain is created, the user must assign domain(s) to the object within the limits of the user's access rights. Domain assignment can be modified at any time.

If a virtual machine management (VMM) domain is tagged as a security domain, the users contained in the security domain can access the correspondingly tagged VMM domain. For example, if a tenant named solar

is tagged with the security domain called sun and a VMM domain is also tagged with the security domain called sun, then users in the solar tenant can access the VMM domain according to their access rights.

Configuring a Custom Role Using the REST API

Procedure

To configure a custom role, send a POST request with XML as in the following example:

Example:

```
<aaaRoleresetToFactory="no"
priv="aaa,access-connectivity-11,access-connectivity-12,access-connectivity-13,access-connectivity-mgmt,
access-connectivity-util,access-equipment,access-protocol-11,access-protocol-12,access-protocol-13,access-protocol-mgmt,
access-protocol-ops,access-protocol-util,access-qos,fabric-connectivity-11,fabric-connectivity-12,
fabric-connectivity-13,fabric-connectivity-mgmt,fabric-connectivity-util,fabric-equipment,
fabric-protocol-11,fabric-protocol-12,fabric-protocol-13,fabric-protocol-mgmt,fabric-protocol-ops,
fabric-protocol-util,nw-svc-device,nw-svc-devshare,nw-svc-policy,ops,tenant-connectivity-11,
tenant-connectivity-12,tenant-connectivity-13,tenant-connectivity-mgmt,tenant-connectivity-util,
tenant-epg,tenant-ext-connectivity-11,tenant-ext-connectivity-12,tenant-ext-connectivity-13,
tenant-ext-connectivity-mgmt,tenant-ext-connectivity-util,tenant-ext-protocol-11,tenant-ext-protocol-12,
tenant-ext-protocol-13,tenant-ext-protocol-mgmt,tenant-ext-protocol-util,tenant-network-profile,
tenant-protocol-11,tenant-protocol-12,tenant-protocol-13,tenant-protocol-mgmt,tenant-protocol-ops,
tenant-protocol-util,tenant-qos,tenant-security,vmm-connectivity,vmm-ep,vmm-policy,vmm-protocol-ops,
vmm-security" ownerTag="" ownerKey="" name="tenant-admin"
dn="uni/userext/role-tenant-admin" descr=""/>
```

Configuring a Local User

In the initial configuration script, the admin account is configured and the admin is the only user when the system starts. The APIC supports a granular, role-based access control system where user accounts can be created with various roles including non-admin users with fewer privileges.

Configuring a Local User Using the REST API

Procedure

Create a local user.

Example:

```
URL: https://apic-ip-address/api/policymgr/mo/uni/userext.xml
POST CONTENT:
<aaaUser name="operations" phone="" pwd="<strong_password>" >
    <aaaUserDomain childAction="" descr="" name="all" rn="userdomain-all" status="">
        <aaaUserRole childAction="" descr="" name="Ops" privType="writePriv"/>
    </aaaUserDomain>
</aaaUser>
```

Configuring a Remote User

Instead of configuring local users, you can point the APIC at the centralized enterprise credential datacenter. The APIC supports Lightweight Directory Access Protocol (LDAP), active directory, RADIUS, and TACACS+.



Note

When an APIC is in minority (disconnected from the cluster), remote logins can fail because the ACI is a distributed system and the user information is distributed across APICS. Local logins, however, continue to work because they are local to the APIC.

To configure a remote user authenticated through an external authentication provider, you must meet the following prerequisites:

- The DNS configuration should have already been resolved with the hostname of the RADIUS server.
- You must configure the management subnet.

Configuring a Remote User Using the REST API

Procedure

Step 1

Create a RADIUS provider.

Example:

```
URL: https://apic-ip-address/api/policymgr/mo/uni/userext/radiusext.xml
POST Content:
<aaaRadiusProvider name="radius-auth-server.org.com" key="test123" />
```

Step 2

Create a login domain.

Example:

```
URL: https://apic-ip-address/api/policymgr/mo/uni/userext.xml
POST Content:
<aaaLoginDomain name="rad"> <aaaDomainAuth realm="radius"/> </aaaLoginDomain>
```

APIC Signature-Based Transactions

About Signature-Based Transactions

The APIC controllers in a Cisco ACI fabric offer different methods to authenticate users.

The primary authentication method uses a username and password and the APIC REST API returns an authentication token that can be used for future access to the APIC. This may be considered insecure in a situation where HTTPS is not available or enabled.

Another form of authentication that is offered utilizes a signature that is calculated for every transaction. The calculation of that signature uses a private key that must be kept secret in a secure location. When the APIC receives a request with a signature rather than a token, the APIC utilizes an X.509 certificate to verify the signature. In signature-based authentication, every transaction to the APIC must have a newly calculated signature. This is not a task that a user should do manually for each transaction. Ideally this function should be utilized by a script or an application that communicates with the APIC. This method is the most secure as it requires an attacker to crack the RSA/DSA key to forge or impersonate the user credentials.

**Note**

Additionally, you must use HTTPS to prevent replay attacks.

Before you can use X.509 certificate-based signatures for authentication, verify that the following pre-requisite tasks are completed:

- 1 Create an X.509 certificate and private key using OpenSSL or a similar tool.
- 2 Create a local user on the APIC. (If a local user is already available, this task is optional).
- 3 Add the X.509 certificate to the local user on the APIC.

Using a Private Key to Calculate a Signature

Before You Begin

You must have the following information available:

- HTTP method - GET, POST, DELETE
- REST API URI being requested, including any query options
- For POST requests, the actual payload being sent to the APIC
- The private key used to generate the X.509 certificate for the user
- The distinguished name for the user X.509 certificate on the APIC

Procedure

-
- Step 1** Concatenate the HTTP method, REST API URI, and payload together in this order and save them to a file.

This concatenated data must be saved to a file for OpenSSL to calculate the signature. In this example, we use a filename of payload.txt. Remember that the private key is in a file called userabc.key.

Example:

GET example:

```
GET http://10.10.10.1/api/class/fvTenant.json?rsp-subtree=children
```

POST example:

```
POST http://10.10.10.1/api/mo/tn-test.json{"fvTenant": {"attributes": {"status": "deleted", "name": "test"}}}
```

- Step 2** Calculate a signature using the private key and the payload file using OpenSSL.

Example:

```
openssl dgst -sha256 -sign userabc.key payload.txt > payload_sig.bin
```

The resulting file has the signature printed on multiple lines.

- Step 3** Strip the signature of the new lines using Bash.

Example:

```
$ tr -d '\n' < payload_sig.base64
P+OTqK0CeAZj17+Gute2R1Ww8OGgtzE0wsLlx8fIXXl4V79Z17
Ou8IdJH9CB4W6CEvdICXqkv3KaQszCIC0+Bn07o3qF//BsIplZmYChD6gCX3f7q
IcjGX+R6HAqGeK7k97cNhXlWEoobFPe/oajtPjOu3tdOjhf/9ujG6Jv6Ro=
```

Note This is the signature that will be sent to the APIC for this specific request. Other requests will require to have their own signatures calculated.

- Step 4** Place the signature inside a string to enable the APIC to verify the signature against the payload. This complete signature is sent to the APIC as a cookie in the header of the request.

Example:

```
APIC-Request-Signature=P+OTqK0CeAZj17+Gute2R1Ww8OGgtzE0wsLlx8f
IXXl4V79Z17Ou8IdJH9CB4W6CEvdICXqkv3KaQszCIC0+Bn07o3qF//BsIplZmYChD6gCX3f
7qIcjGX+R6HAqGeK7k97cNhXlWEoobFPe/oajtPjOu3tdOjhf/9ujG6Jv6Ro=
APIC-Certificate-Algorithm=v1.0; APIC-Certificate-Fingerprint=fingerprint;
APIC-Certificate-DN=uni/userext/user-userabc/usercert-userabc.crt
```

Note The DN used here must match the DN of the user certified object containing the x509 certificate in the next step.

- Step 5** Use the CertSession class in the Python SDK to communicate with an APIC using signatures. The following script is an example of how to use the CertSession class in the ACI Python SDK to make requests to an APIC using signatures.

Example:

```
#!/usr/bin/env python
# It is assumed the user has the X.509 certificate already added to
# their local user configuration on the APIC
from cobra.mit.session import CertSession
from cobra.mit.access import MoDirectory

def readFile(fileName=None, mode="r"):
    if fileName is None:
        return ""
    fileData = ""
    with open(fileName, mode) as aFile:
        fileData = aFile.read()
    return fileData

pkey = readFile("/tmp/userabc.key")
csession = CertSession("https://ApicIPOrHostname/",
                      "uni/userext/user-userabc/usercert-userabc.crt", pkey)
```

```

modir = MoDirectory(csession)
resp = modir.lookupByDn('uni/fabric')
print resp.dn
# End of script

```

Note The DN used in the earlier step must match the DN of the user certified object containing the x509 certificate in this step.

Guidelines and Limitations

Follow these guidelines and limitations:

- Local users are supported. Remote AAA users are not supported.
- The APIC GUI does not support the certificate authentication method.
- WebSockets and eventchannels do not work for X.509 requests.
- Certificates signed by a third party are not supported. Use a self-signed certificate.

Creating a Local User and Adding a User Certificate Using the REST API

Procedure

Create a local user and add a user certificate.

Example:

```

method: POST
url: http://apic/api/node/mo/uni/userext/user-userabc.json
payload:
{
    "aaaUser": {
        "attributes": {
            "name": "userabc",
            "firstName": "Adam",
            "lastName": "BC",
            "phone": "408-525-4766",
            "email": "userabc@cisco.com",
        },
        "children": [
            "aaaUserCert": {
                "attributes": {
                    "name": "userabc.crt",
                    "data": "-----BEGIN CERTIFICATE-----\nMIICjjCCAfegAwIBAgIJAMQnbE<snipped content> ==\n-----END CERTIFICATE-----",
                },
                "children": []
            },
            "aaaUserDomain": {
                "attributes": {
                    "name": "all",
                },
                "children": [
                    "aaaUserRole": {
                        "attributes": {
                            "name": "aaa",
                            "privType": "writePriv",
                        },
                        "children": []
                    }
                ]
            }
        ]
    }
}

```

```

        }
    },
    "aaaUserRole": {
        "attributes": {
            "name": "access-admin",
            "privType": "writePriv",
        },
        "children": []
    },
    {
        "aaaUserRole": {
            "attributes": {
                "name": "admin",
                "privType": "writePriv",
            },
            "children": []
        }
    },
    {
        "aaaUserRole": {
            "attributes": {
                "name": "fabric-admin",
                "privType": "writePriv",
            },
            "children": []
        }
    },
    {
        "aaaUserRole": {
            "attributes": {
                "name": "nw-svc-admin",
                "privType": "writePriv",
            },
            "children": []
        }
    },
    {
        "aaaUserRole": {
            "attributes": {
                "name": "ops",
                "privType": "writePriv",
            },
            "children": []
        }
    },
    {
        "aaaUserRole": {
            "attributes": {
                "name": "read-all",
                "privType": "writePriv",
            },
            "children": []
        }
    },
    {
        "aaaUserRole": {
            "attributes": {
                "name": "tenant-admin",
                "privType": "writePriv",
            },
            "children": []
        }
    },
    {
        "aaaUserRole": {
            "attributes": {
                "name": "tenant-ext-admin",
                "privType": "writePriv",
            },
            "children": []
        }
    },
    {
        "aaaUserRole": {
            "attributes": {
                "name": "vmm-admin",
                "privType": "writePriv",
            },
            "children": []
        }
    }
}

```

```
        }
      }]
    }
}
```



CHAPTER 4

Common Tenant Tasks

- [Common Tenant Tasks, page 81](#)

Common Tenant Tasks

Tenants Overview

- A tenant contains policies that enable qualified users domain-based access control. Qualified users can access privileges such as tenant administration and networking administration.
- A user requires read/write privileges for accessing and configuring policies in a domain. A tenant user can have specific privileges into one or more domains.
- In a multitenancy environment, a tenant provides group user access privileges so that resources are isolated from one another (such as for endpoint groups and networking). These privileges also enable different users to manage different tenants.

Tenant Creation

A tenant contains primary elements such as filters, contracts, bridge domains, and application profiles that you can create after you first create a tenant.

Adding a Tenant

A tenant is a policy owner in the virtual fabric. A tenant can be either a private or a shared entity. For example, you can create a securely partitioned private tenant or a tenant with contexts and bridge domains shared by other tenants. A shared type of tenant is typically named common, default, or infra.

In the management information model, a tenant is represented by a managed object (MO) of class fv:Tenant. According to the *Cisco APIC Management Information Model Reference*, an object of the fv:Tenant class is a child of the policy resolution universe (uni) class and has a distinguished name (DN) format of `uni/tn-[name]`.



Note You can only add one tenant at a time.

The following examples show how to add a new tenant named ExampleCorp using XML and JSON.

Example: Using the JSON API to Add a Tenant

To create a new tenant, you must specify the class and sufficient naming information, either in the message body or in the URI.

To create a new tenant using the JSON API, send this HTTP POST message:

```
POST https://apic-ip-address/api/mo/uni.json
{
  "fvTenant" : {
    "attributes" : {
      "name" : "ExampleCorp"
    }
  }
}
```

Alternatively, you can name the tenant in the URI, as in this example:

```
POST https://apic-ip-address/api/mo/uni/tn-ExampleCorp.json
{
  "fvTenant" : {
    "attributes" : {
    }
  }
}
```

If a response is requested (by appending `?rsp-subtree=modified` to the POST URI), a successful operation returns the following response body:

```
{
  "imdata" :
  [
    {
      "fvTenant" : {
        "attributes" : {
          "instanceId" : "0:0",
          "childAction" : "deleteNonPresent",
          "dn" : "uni/tn-ExampleCorp",
          "lcOwn" : "local",
          "name" : "ExampleCorp",
          "replTs" : "never",
          "rn" : "",
          "status" : "created"
        }
      }
    }
  ]
}
```

To delete the tenant, send this HTTP DELETE message:

```
DELETE https://apic-ip-address/api/mo/uni/tn-ExampleCorp.json
```

Alternatively, you can send an HTTP POST message with sufficient naming information and with "status": "deleted" in the fv:Tenant attributes, as in this example:

```
POST https://apic-ip-address/api/mo/uni.json
{
  "fvTenant" : {
    "attributes" : {
      "name" : "ExampleCorp",
      "status" : "deleted"
    }
  }
}
```

Example: Using the XML API to Add a Tenant

To create a new tenant, you must specify the class and sufficient naming information, either in the message body or in the URI.

To create a new tenant named ExampleCorp using the XML API, send this HTTP POST message:

```
POST https://apic-ip-address/api/mo/uni.xml
<fvTenant name="ExampleCorp"/>
```

Alternatively, you can name the tenant in the URI, as in this example:

```
POST https://apic-ip-address/api/mo/uni/tn-ExampleCorp.xml
<fvTenant />
```

If a response is requested (by appending ?rsp-subtree=modified to the POST URI), a successful operation returns the following response body:

```
<imdata>
  <fvTenant
    instanceId="0:0"
    childAction="deleteNonPresent"
    dn="uni/tn-ExampleCorp"
    lcOwn="local"
    name="ExampleCorp"
    replTs="never"
    rn=""
    status="created"
  />
</imdata>
```

To delete the tenant, send this HTTP DELETE message:

```
DELETE https://apic-ip-address/api/mo/uni/tn-ExampleCorp.xml
```

Alternatively, you can send an HTTP POST message with sufficient naming information and with status="deleted" in the fv:Tenant attributes, as in this example:

```
POST https://apic-ip-address/api/mo/uni.xml
<fvTenant name="ExampleCorp" status="deleted"/>
```




CHAPTER 5

Managing Layer 2 Networking

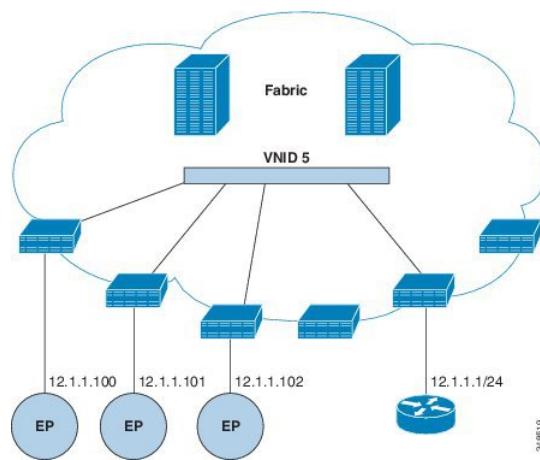
- Tenant External Bridged Networks, page 85
- Ports, page 86
- Creating a Port Channel Policy Using the REST API, page 89

Tenant External Bridged Networks

Bridged Interface to an External Router

As shown in the figure below, when the leaf switch interface is configured as a bridged interface, the default gateway for the tenant VNID is the external router.

Figure 4: Bridged External Router



The ACI fabric is unaware of the presence of the external router and the APIC statically assigns the leaf switch interface to its EPG.

VRF and Bridge Domains

You can create and specify a VRF and a bridge domain for the tenant. The defined bridge domain element subnets reference a corresponding Layer 3 context.

For details about enabling IPv6 Neighbor Discovery see the related KB article, *KB: Creating a Tenant, VRF, and Bridge Domain with IPv6 Neighbor Discovery*.

Creating a Tenant, VRF, and Bridge Domain Using the REST API

Procedure

Step 1 Create a tenant.

Example:

```
POST https://apic-ip-address/api/mo/uni.xml
<fvTenant name="ExampleCorp"/>
```

When the POST succeeds, you see the object that you created in the output.

Step 2 Create a VRF and bridge domain.

Note The Gateway Address can be an IPv4 or an IPv6 address. For more about details IPv6 gateway address, see the related KB article, *KB: Creating a Tenant, VRF, and Bridge Domain with IPv6 Neighbor Discovery*.

Example:

URL for POST: <https://apic-ip-address/api/mo/uni/tn-ExampleCorp.xml>

```
<fvTenant name="ExampleCorp">
  <fvCtx name="pvn1"/>
  <fvBD name="bd1">
    <fvRsCtx tnFvCtxName="pvn1"/>
    <fvSubnet ip="10.10.100.1/24"/>
  </fvBD>
</fvTenant>
```

Note If you have a public subnet when you configure the routed outside, you must associate the bridge domain with the outside configuration.

Ports

Statically Deploying an EPG on a Specific Port

This topic provides a typical example of how to statically deploy an EPG on a specific port when using Cisco APIC.

Deploying an EPG on a Specific Port with APIC Using the REST API

Before You Begin

The tenant where you deploy the EPG is created.

Procedure

Deploy an EPG on a specific port.

Example:

```
<fvTenant name=<tenant_name> dn="uni/tn-test1" >
  <fvCtx name=<network_name> pcEnfPref="enforced" knwMcastAct="permit"/>
  <fvBD name=<bridge_domain_name> unkMcastAct="flood" >
    <fvRsCtx tnFvCtxName=<network_name>/>
  </fvBD>
  <fvAp name=<application_profile> >
    <fvAEPg name=<epg_name> >
      <fvRsPathAtt tDn="topology/pod-1/paths-1017/pathEp-[eth1/13]" mode="regular"
instrImedcy="immediate" encap="vlan-20"/>
    </fvAEPg>
  </fvAp>
</fvTenant>
```

Creating Domains, Attach Entity Profiles, and VLANs to Deploy an EPG on a Specific Port

This topic provides a typical example of how to create physical domains, Attach Entity Profiles (AEP), and VLANs that are mandatory to deploy an EPG on a specific port.



Note

All endpoint groups (EPGs) require a domain. Interface policy groups must also be associated with Attach Entity Profile (AEP), and the AEP must be associated with a domain, if the AEP and EPG have to be in same domain. Based on the association of EPGs to domains and of interface policy groups to domains, the ports and VLANs that the EPG uses are validated. The following domain types associate with EPGs:

- Application EPGs
- Layer 3 external outside network instance EPGs
- Layer 2 external outside network instance EPGs
- Management EPGs for out-of-band and in-band access

The APIC checks if an EPG is associated with one or more of these types of domains. If the EPG is not associated, the system accepts the configuration but raises a fault. The deployed configuration may not function properly if the domain association is not valid. For example, if the VLAN encapsulation is not valid for use with the EPG, the deployed configuration may not function properly.

Creating AEP, Domains, and VLANs to Deploy an EPG on a Specific Port Using the REST API

Before You Begin

- The tenant where you deploy the EPG is already created.

- An EPG is statically deployed on a specific port.

Procedure

Step 1 Create the interface profile, switch profile and the Attach Entity Profile (AEP).

Example:

```
<infraInfra>
    <infraNodeP name=<switch_profile_name> dn="uni/infra/nprof-<switch_profile_name>">
        <infraLeafS name="SwitchSelector" descr="" type="range">
            <infraNodeBlk name="nodeBlk1" descr="" to_="1019" from_="1019"/>
        </infraLeafS>
        <infraRsAccPortP tDn="uni/infra/accportprof-<interface_profile_name>"/>
    </infraNodeP>

    <infraAccPortP name=<interface_profile_name> dn="uni/infra/accportprof-<interface_profile_name>">
        <infraHPortS name="portSelector" type="range">
            <infraRsAccBaseGrp tDn="uni/infra/funcprof/accportgrp-<port_group_name>" fexId="101"/>
            <infraPortBlk name="block2" toPort="13" toCard="1" fromPort="11" fromCard="1"/>
        </infraHPortS>
    </infraAccPortP>

    <infraAccPortGrp name=<port_group_name> dn="uni/infra/funcprof/accportgrp-<port_group_name>">
        <infraRsAttEntP tDn="uni/infra/attentp-<attach_entity_profile_name>"/>
        <infraRsHIfPol tnFabricHIfPolName="1GHifPol"/>
    </infraAccPortGrp>

    <infraAttEntityP name=<attach_entity_profile_name> dn="uni/infra/attentp-<attach_entity_profile_name>">
        <infraRsDomP tDn="uni/phys-<physical_domain_name>"/>
    </infraAttEntityP>
</infraInfra>
```

Step 2 Create a domain.

Example:

```
<physDomP name=<physical_domain_name> dn="uni/phys-<physical_domain_name>">
    <infraRsVlanNs tDn="uni/infra/vlanns-[<vlan_pool_name>]-static"/>
</physDomP>
```

Step 3 Create a VLAN range.

Example:

```
<fvnsVlanInstP name=<vlan_pool_name> dn="uni/infra/vlanns-[<vlan_pool_name>]-static" allocMode="static">
    <fvnsEncapBlk name="" descr="" to="vlan-25" from="vlan-10"/>
</fvnsVlanInstP>
```

Step 4 Associate the EPG with the domain.

Example:

```
<fvTenant name=<tenant_name> dn="uni/tn->
    <fvAEPg prio="unspecified" name=<epg_name> matchT="AtleastOne" dn="uni/tn-test1/ap-AP1/epg-<epg_name>" descr="">
        <fvRsDomAtt tDn="uni/phys-<physical_domain_name>" instrImedcy="immediate" resImedcy="immediate"/>
```

```
</fvAEPg>  
</fvTenant>
```

Creating a Port Channel Policy Using the REST API

The following example REST request creates a Port Channel policy:

```
<lacpLagPol childAction="" ctrl="fast-sel-hot-stdby,graceful-conv,susp-individual"  
descr="" dn="uni/infra/lacplagp-LACP-Active" lcOwn="local" maxLinks="16" minLinks="1"  
modTs="2015-02-24T11:58:36.547-08:00" mode="active" name="LACP-Active" ownerKey=""  
ownerTag="" status="" uid="8131">  
  <lacpRtLacpPol childAction="" lcOwn="local" modTs="2015-02-24T14:59:11.154-08:00"  
  rn="rtInfraLacpPol-[uni/infra/funcprof/accbundle-ACI-VPC-IPG]" status=""  
  tCl="infraAccBndlGrp" tDn="uni/infra/funcprof/accbundle-ACI-VPC-IPG"/>  
</lacpLagPol>
```




CHAPTER 6

Managing Layer 3 Networking

- [Configuring External Connectivity for Tenants, page 91](#)
- [Configuring a Tenant Layer 3 Outside Network Connection Overview, page 91](#)
- [Configuring Layer 3 Outside for Tenant Networks Using the REST API, page 91](#)

Configuring External Connectivity for Tenants

Before you can distribute the static route to the other leaf switches on the Application Centric Infrastructure (ACI) fabric, a multiprotocol BGP (MP-BGP) process must first be operating, and the spine switches must be configured as BGP route reflectors.

To integrate the ACI fabric into an external routed network, you can configure Open Shortest Path First (OSPF) for management tenant Layer 3 connectivity.

Configuring a Tenant Layer 3 Outside Network Connection Overview

This topic provides a typical example of how to configure a Layer 3 Outside for tenant networks when using Cisco APIC.

Configuring Layer 3 Outside for Tenant Networks Using the REST API

The external routed network configured in the example can also be extended to support IPv4. Both IPv4 and IPv6 routes can be advertised to and learned from the external routed network.

Before You Begin

- The tenant, private network, and bridge domain are created.
- The external routed domain is created.

Procedure

Configure L3 Outside for tenant networks and associate the bridge domain with the Layer3 Outside.

Example:

```

<fvTenant name='TenantName'>
  <l3extOut name="L3Out1" enforceRtctrl="import,export">
    <l3extRsL3DomAtt tDn="uni/l3dom-l3DomP"/>
    <l3extLNodeP name="LNodeP1" >
      <l3extRsNodeL3OutAtt rtrId="1.2.3.4" tDn="topology/pod-1/node-101">
        <l3extLoopBackIfP addr="10.10.11.1" />
        <l3extLoopBackIfP addr="2000::3" />
      </l3extRsNodeL3OutAtt>
      <l3extLIfP name="IFP1" >
        <l3extRsPathL3OutAtt addr="10.11.12.10/24" ifInstT="l3-port"
          tDn="topology/pod-1/paths-103/pathep-[eth1/17]" />
        </l3extLIfP>
      <l3extLIfP name="IFP2" >
        <l3extRsPathL3OutAtt addr="2001::3/64" ifInstT="l3-port"
          tDn="topology/pod-1/paths-103/pathep-[eth1/17]" />
        </l3extLIfP>
      </l3extLNodeP>
      <l3extRsEctx tnFvCtxName="VRF1"/>
      <l3extInstP name="InstP1" >
        <l3extSubnet ip="192.168.1.0/24" scope="import-security" aggregate="" />
        <l3extSubnet ip="0.0.0.0/0" scope="export-rtctrl,import-rtctrl,import-security"
          aggregate="export-rtctrl,import-rtctrl"/>
        <l3extSubnet ip="192.168.2.0/24" scope="export-rtctrl" aggregate="" />
        <l3extSubnet ip="::/0" scope="import-rtctrl,import-security"
          aggregate="import-rtctrl"/>
        <l3extSubnet ip="2001:17a::/64" scope="export-rtctrl" aggregate="" />
      </l3extInstP>
    </l3extOut>
  </fvTenant>

```

Note The "enforceRtctrl=import" is not applicable for EIGRP.



CHAPTER 7

Monitoring Using the REST API

- [About Monitoring Using the REST API, page 93](#)
- [APIC, page 93](#)
- [Fabric, page 95](#)
- [Switches, page 96](#)

About Monitoring Using the REST API

Monitoring APIC Using the REST API

Proactive monitoring is an important piece of the network administrator's job but is often neglected because solving urgent problems in the network usually takes priority. However, the Application Policy Infrastructure Controller (APIC) will save network administrators time and frustration because it makes it easy to gather statistics and perform analyses. Because statistics are gathered automatically and policies are used and can be re-used in other places, human effort and error are minimized.

The following examples using the REST API can be used to drill into APIC fabric and switch components.

APIC

Monitoring APIC CPU and Memory Usage Using the REST API

The easiest way to quickly verify the health of the controllers is the APIC. Controllers provide information regarding the current status of CPU and memory utilization by creating instances of the *procEntity* class. The *procEntity* is a container of processes in the system. This object holds detailed information about various processes running on the APIC. The *procEntity* objects contain the following useful properties:

- *cpuPct*—CPU utilization
- *maxMemAlloc*—The maximum memory allocated for the system
- *memFree*—The maximum amount of available memory for the system

Procedure

Retrieve information about CPU and memory usage using the following REST API call:

Example:

`https://apic-ip-address/api/node/class/procEntity.xml?`

Monitoring APIC Disk Utilization Using the REST API

There are several disks and file systems present on the APIC. The REST API provides ready access to disk space utilization of all partitions on the system and can be used for monitoring this information.

Procedure

Monitor the disk and file systems on the APIC, by sending a REST API post, such as the following:

Example:

`https://apic-ip-address/api/node/class/eqptStorage.xml?`

Monitoring Physical Interface Statistics and Link State Using the REST API

You can use the REST API interface to poll for interface statistics. Several counters are available (for example, RX/TX, input/output / duplex, 30 second rates, 5 minute rate, unicast packets, multicast packets). Using the parent managed object, the children can be derived from it. To do this, you must have a good understanding of the object model and be able to navigate through the model to obtain the information desired using the example below.

Procedure

- Step 1** Use the following base API call to get physical interface statistics:

Example:

`https://apic-ip-address/api/node/mo/topology/pod-1/node-101/sys/phys-[eth1/1].json`

- Step 2** To determine the total ingress bytes on Leaf 101 port Eth1/1, you can issue the following API call:

Example:

`/topology/pod-1/node-101/sys/phys-[eth1/1].json`

- Step 3** Visore allows you to dig deeper into the hierarchical tree. From the prior command, the operator can see children of the interface object, such as ingress and egress bytes. The child objects include the following:

Example:

`/topology/pod-1/node-101/sys/phys-[eth1/1]/dbgEtherStats`

Fabric

Monitoring LLDP and CDP Neighbor Status Using the REST API

The APIC enables you to determine all LLDP or CDP neighbors in a fabric, using the REST API.

Procedure

- Step 1** To determine the LLDP neighbors, send a POST such as the following:

Example:

```
https://apic-ip-address/api/node/mo/topology/pod-1/node-101/sys/lldp/inst/if-[eth1/1]
```

- Step 2** To determine the CDP neighbors, send a POST such as the following:

Example:

```
https://apic-ip-address/api/node/mo/topology/pod-1/node-101/sys/cdp/inst/if-[eth1/1]
```

Monitoring Physical and Bond Interfaces Using the REST API

The APIC uses a bonded interface that is typically dual-homed to two leaf switches for connectivity to the Cisco ACI fabric. It also can use a bonded interface that can be dual-homed to the out-of-band management network.

- *Bond0* is the bond interface used to connect to the fabric itself (to connect to leaf switches that connect into the fabric).
- *Bond1* is the bond interface used to connect to the out-of-band segment (to connect to an OOB segment that allows setup of the APIC itself).

The bond interfaces rely on underlying physical interfaces. It is important to note that the REST API provides link information for both the physical and logical bond interfaces.

Procedure

Collect information about both the bond interfaces by sending a POST request such as the following example:

Example:

```
https://apic-ip-address/api/node/mo/topology/pod-1/node-1/sys.json?querytarget=subtree&target-subtree-class=l3EncRtdIf
```

Monitoring EPG-Level Statistics Using the REST API

To monitor network-related information for an application, you can investigate the aggregate amount of traffic to a specific tier. For example, you can monitor the amount of traffic to the web tier of a given EPG application with the REST API.

Procedure

To monitor the traffic for a new project for the epg-web-epg, send a POST request such as the following example:

Example:

```
https://apic-ip-address/api/node/mo/uni/tn-newproject/ap-app1/epg-web-epg.xml?querytarget= self&rsp-subtree-inclu
```

Switches

Monitoring Switch CPU Utilization Using the REST API

Spine and leaf switch CPU utilization can be monitored using the following classes, based on the desired timescale and granularity:

- **proc:SysCPU5min**—A class that represents the most current statistics for system CPU in a 5-minute sampling interval. This class updates every 10 seconds.
- **proc:SysCPU15min**—A class that represents the most current statistics for system CPU in a 15-minute sampling interval. This class updates every 5 minutes.
- **proc:SysCPU1h**—A class that represents the most current statistics for system CPU in a 1-hour sampling interval. This class updates every 15 minutes.
- **proc:SysCPU1d**—A class that represents the most current statistics for system CPU in a 1-day sampling interval. This class updates every hour.
- **proc:SysCPU1w**—A class that represents the most current statistics for system CPU in a 1-week sampling interval. This class updates every day.
- **proc:SysCPU1mo**—A class that represents the most current statistics for system CPU in a 1-month sampling interval. This class updates every day.
- **proc:SysCPU1qtr**—A class that represents the most current statistics for system CPU in a 1-quarter sampling interval. This class updates every day.
- **proc:SysCPU1year**—A class that represents the most current statistics for system CPU in a 1-year sampling interval. This class updates every day.

The following example shows how to use these classes for monitoring:

Procedure

To view the average CPU utilization of all of the fabric switches over the last day, use XML such as in the following example:

Example:

```
https://apic-ip-address//api/node/class/procSysCPU1d.xml?
```

Monitoring Switch Fan Status Using the REST API

The following REST API call(s) and their child objects can be used to monitor the state of the fans on a leaf switch (note that there are 3 slots on this particular switch).

Procedure

To retrieve the status of the fan trays on the leaf switches, use XML such as the following example:

Example:

```
https://apic-ip-address/api/node/mo/topology/pod-1/node-101/sys/ch/ftslot-1
https://apic-ip-address/api/node/mo/topology/pod-1/node-101/sys/ch/ftslot-2
https://apic-ip-address/api/node/mo/topology/pod-1/node-101/sys/ch/ftslot-3
```

Monitoring Switch Memory Utilization Using the REST API

Spine and leaf switch memory utilization can be monitored using the following classes, based on the desired timescale and granularity:

- **proc:SysMem5min**—A class that represents the most current statistics for system memory in a 5-minute sampling interval. This class updates every 10 seconds.
- **proc:SysMem15min**—A class that represents the most current statistics for system memory in a 15-minute sampling interval. This class updates every 5 minutes.
- **proc:SysMem1h**—A class that represents the most current statistics for system memory in a 1-hour sampling interval. This class updates every 15 minutes.
- **proc:SysMem1d**—A class that represents the most current statistics for system memory in a 1-day sampling interval. This class updates every hour.
- **proc:SysMem1w**—A class that represents the most current statistics for system memory in a 1-week sampling interval. This class updates every day.
- **proc:SysMem1mo**—A class that represents the most current statistics for system memory in a 1-month sampling interval. This class updates every day.
- **proc:SysMem1qtr**—A class that represents the most current statistics for system memory in a 1-quarter sampling interval. This class updates every day.
- **proc:SysMem1year**—A class that represents the most current statistics for system memory in a 1-year sampling interval. This class updates every day.

The following example shows how to use one of the classes:

Procedure

To monitor memory over the last day, use the following REST call:

Example:

```
https://apic-ip-address/api/node/class/procSysMem1d.xml?
```

Monitoring Switch Module Status Using the REST API

Even though the leaves are considered fixed switches, they have a supervisor component that refers to the CPU complex. From a forwarding perspective, there are two data-plane components: the NFE (Network Forwarding Engine) ASIC, which provides the front panel ports; and the ALE or ALE2 (Application Leaf Engine) ASIC—depending on the generation of switch hardware—which provides uplink connectivity to the spines. The following REST API example can be used to determine the status of the modules in the switch:

Procedure

To monitor the state of the supervisor and the module, use a REST API call such as the following:

Example:

```
https://apic-ip-address/api/node/mo/topology/pod-1/node-101/sys/ch/supslot-1/sup  
https://apic-ip-address/api/node/mo/topology/pod-1/node-101/sys/ch/lcslot-1/lc
```

Monitoring Switch Power Supply Status Using the REST API

You can use the REST API to retrieve the status of the power supplies on the leaf switches.

Procedure

To monitor the state of the power supplies on a leaf switch, use XML such as the following example:

Example:

```
https://apic-ip-address/api/node/mo/topology/pod-1/node-101/sys/ch/psuslot-1  
https://apic-ip-address/api/node/mo/topology/pod-1/node-101/sys/ch/psuslot-2
```

Note that there are 2 power supplies on this particular switch.

Monitoring Switch Inventory Using the REST API

You can use the REST API to retrieve switch hardware information such as the model and serial numbers.

Procedure

To retrieve switch hardware information, use the REST API as shown in the following example:

Example:

```
https://apic-ip-address/api/node/mo/topology/pod-1.json?query-target=children&target-subtree-class=fabricNode
```



CHAPTER 8

Troubleshooting Using the REST API

- [Collecting and Exporting Technical Support Information, page 99](#)
- [Troubleshooting Using Atomic Counters, page 100](#)
- [Troubleshooting Using Faults, page 103](#)
- [Statistics, page 105](#)
- [Recovering a Disconnected Leaf, page 106](#)
- [Troubleshooting Contracts and Taboo Contracts with Permit and Deny Logging, page 107](#)
- [Troubleshooting Using Digital Optical Monitoring Statistics, page 108](#)
- [Troubleshooting Using Port Tracking, page 109](#)
- [Removing Unwanted _ui_ Objects, page 110](#)

Collecting and Exporting Technical Support Information

About Exporting Files

An administrator can configure export policies in the APIC to export statistics, technical support collections, faults and events, to process core files and debug data from the fabric (the APIC as well as the switch) to any external host. The exports can be in a variety of formats, including XML, JSON, web sockets, secure copy protocol (SCP), or HTTP. You can subscribe to exports in streaming, periodic, or on-demand formats.

An administrator can configure policy details such as the transfer protocol, compression algorithm, and frequency of transfer. Policies can be configured by users who are authenticated using AAA. A security mechanism for the actual transfer is based on a username and password. Internally, a policy element handles the triggering of data.

Sending an On-Demand TechSupport File Using the REST API

Procedure

- Step 1** Set the remote destination for a technical support file using the REST API, by sending a POST with XML such as the following example:

Example:

```
<fileRemotePath userName="" remotePort="22" remotePath="" protocol="sftp" name="ToSupport"
host="192.168.200.2"
dn="uni/fabric/path-ToSupport" descr="">
<fileRsARemoteHostToEpg tDn="uni/tn-mgmt/mgmtp-default/oob-default"/>
</fileRemotePath>
```

- Step 2** Generate an on-demand technical support file using the REST API by sending a POST with XML such as the following:

Example:

```
<dbgexpTechSupOnD upgradeLogs="no" startTime="unspecified" name="Tech_Support_9-20-16"
exportToController="no"
endTime="unspecified" dn="uni/fabric/tsod-Tech_Support_9-20-16" descr="" compression="gzip"
category="forwarding" adminSt="untriggered">
<dbgexpRsExportDest tDn="uni/fabric/path-ToSupport"/>
<dbgexpRsTsSrc tDn="topology/pod-1/node-102/sys"/>
<dbgexpRsTsSrc tDn="topology/pod-1/node-103/sys"/>
<dbgexpRsTsSrc tDn="topology/pod-1/node-101/sys"/>
<dbgexpRsData tDn="uni/fabric/tscont"/>
</dbgexpTechSupOnD>
```

Troubleshooting Using Atomic Counters

Atomic Counters

Atomic Counters are useful for troubleshooting connectivity between endpoints, EPGs, or an application within the fabric. A user reporting application may be experiencing slowness, or atomic counters may be needed for monitoring any traffic loss between two endpoints. One capability provided by atomic counters is the ability to place a trouble ticket into a proactive monitoring mode, for example when the problem is intermittent, and not necessarily happening at the time the operator is actively working the ticket.

Atomic counters can help detect packet loss in the fabric and allow the quick isolation of the source of connectivity issues. Atomic counters require NTP to be enabled on the fabric.

Leaf-to-leaf (TEP to TEP) atomic counters can provide the following:

- Counts of drops, admits, and excess packets

- Short-term data collection such as the last 30 seconds, and long-term data collection such as 5 minutes, 15 minutes, or more
- A breakdown of per-spine traffic (available when the number of TEPs, leaf or VPC, is less than 64)
- Ongoing monitoring

Leaf-to-leaf (TEP to TEP) atomic counters are cumulative and cannot be cleared. However, because 30 second atomic counters reset at 30 second intervals, they can be used to isolate intermittent or recurring problems.

Tenant atomic counters can provide the following:

- Application-specific counters for traffic across the fabric, including drops, admits, and excess packets
- Modes include the following:
 - Endpoint to endpoint MAC address, or endpoint to endpoint IP address. Note that a single target endpoint could have multiple IP addresses associated with it.
 - EPG to EPG with optional drill down
 - EPG to endpoint
 - EPG to * (any)
 - Endpoint to external IP address



Note

Atomic counters track the amount packets of between the two endpoints and use this as a measurement. They do not take into account drops or error counters in a hardware level.

Dropped packets are calculated when there are less packets received by the destination than transmitted by the source.

Excess packets are calculated when there are more packets received by the destination than transmitted by the source.

Enabling Atomic Counters

To enable using atomic counters to detect drops and misrouting in the fabric and enable quick debugging and isolation of application connectivity issues, create one or more tenant atomic counter policies, which can be one of the following types:

- EP_to_EP—Endpoint to endpoint (**dbgacEpToEp**)
- EP_to_EPG—Endpoint to endpoint group (**dbgacEpToEpg**)
- EP_to_Ext—Endpoint to external IP address (**dbgacEpToExt**)
- EPG_to_EP—Endpoint group to endpoint(**dbgacEpgToEp**)
- EPG_to_EPG—Endpoint group to endpoint group (**dbgacEpgToEpg**)
- EPG_to_IP—Endpoint group to IP address (**dbgacEpgToIp**)
- Ext_to_EP—External IP address to endpoint (**dbgacExtToEp**)
- IP_to_EPG—IP address to endpoint group (**dbgacIpToEpg**)

- Any_to_EP—Any to endpoint (**dbgacAnyToEp**)
- EP_to_Any—Endpoint to any (**dbgacEpToAny**)

Procedure

- Step 1** To create an EP_to_EP policy using the REST API, use XML such as the following example:

Example:

```
<dbgacEpToEp name="EP_to_EP_Policy" ownerTag="" ownerKey=""
dn="uni/tn-Tenant64/acEpToEp-EP_to_EP_Policy" descr="" adminSt="enabled">
<dbgacFilter name="EP_to_EP_Filter" ownerTag="" ownerKey="" descr=""
srcPort="https" prot="tcp" dstPort="https"/>
</dbgacEpToEp>
```

- Step 2** To create an EP_to_EPG policy using the REST API, use XML such as the following example:

Example:

```
<dbgacEpToEpg name="EP_to_EPG_Pol" ownerTag="" ownerKey=""
dn="uni/tn-Tenant64/epToEpg-EP_to_EPG_Pol" descr="" adminSt="enabled">
<dbgacFilter name="EP_to_EPG_Filter" ownerTag="" ownerKey="" descr=""
srcPort="http" prot="tcp" dstPort="http"/>
<dbgacRsToAbsEpg tDn="uni/tn-Tenant64/ap-VRF64_app_prof/epg-EPG64"/>
</dbgacEpToEpg>
```

Troubleshooting Using Atomic Counters with the REST API

Procedure

- Step 1** To get a list of the endpoint-to-endpoint atomic counters deployed within the fabric and the associated details such as dropped packet statistics and packet counts, use the **dbgEpToEpTsIt** class in XML such as the following example:

Example:

<https://apic-ip-address/api/node/class/dbgEpToEpRsIt.xml>

- Step 2** To get a list of external IP-to-endpoint atomic counters and the associated details, use the **dbgacExtToEp** class in XML such as the following example:

Example:

<https://apic-ip-address/api/node/class/dbgExtToEpRsIt.xml>

Troubleshooting Using Faults

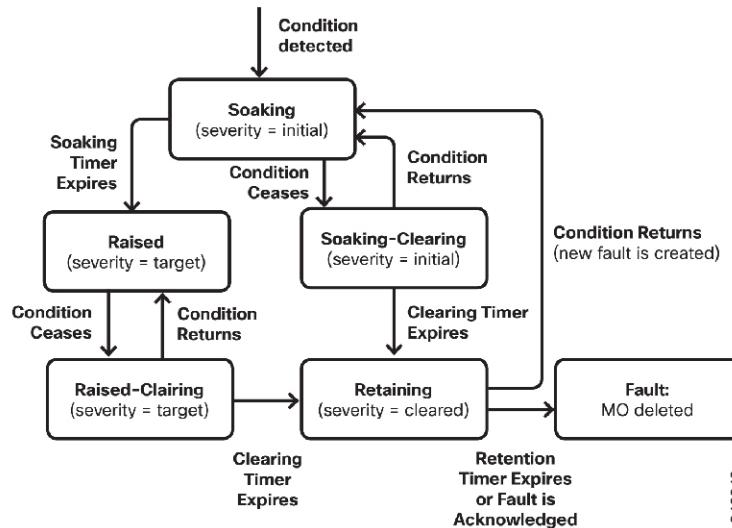
Understanding APIC Faults

From a management point of view we look at the Application Policy Infrastructure Controller (APIC) from two perspectives:

- Policy Controller - Where all fabric configuration is created, managed and applied. It maintains a comprehensive, up-to-date run-time representation of the administrative or configured state.
- Telemetry device - All devices (Fabric Switches, Virtual Switches, integrated Layer 4 to Layer 7 devices) in an Cisco Application Centric Infrastructure (ACI) fabric report faults, events and statistics to the APIC.

Faults, events, and statistics in the ACI fabric are represented as a collection of Managed Objects (MOs) within the overall ACI Object Model/Management Information Tree (MIT). All objects within ACI can be queried, including faults. In this model, a fault is represented as a mutable, stateful, and persistent MO.

Figure 5: Fault Lifecycle



When a specific condition occurs, such as a component failure or an alarm, the system creates a fault MO as a child object to the MO that is primarily associated with the fault. For a fault object class, the fault conditions are defined by the fault rules of the parent object class. Fault MOs appear as regular MOs in MIT; they have a parent, a DN, RN, and so on. The Fault "code" is an alphanumerical string in the form **FXXX**. For more information about fault codes, see the *Cisco APIC Faults, Events, and System Messages Management Guide*.

Troubleshooting Using Faults with the REST API

MOs can be queried by class and DN, with property filters, pagination, and so on.

In most cases, a fault MO is automatically created, escalated, de-escalated, and deleted by the system as specific conditions are detected. There can be at most one fault with a given code under an MO. If the same condition is detected multiple times while the corresponding fault MO is active, no additional instances of the fault MO are created. For example, if the **same condition** is detected multiple times for the **same affected object**, only **one fault** is raised while a counter for the recurrence of that fault will be incremented.

A fault MO remains in the system **until the fault condition is cleared**. For a fault to be removed, the condition raising the fault must be cleared, whether by configuration or a change in the run time state of the fabric. An exception to this is if the fault is in the cleared or retained state, in which case the fault can be deleted by the user by acknowledging it.

Severity provides an indication of the estimated impact of the condition on the capability of the system or component to provide service.

Possible values are:

- Warning (possibly no impact)
- Minor
- Major
- Critical (system or component completely unusable)

The creation of a fault MO can be triggered by internal processes such as:

- Finite state machine (FSM) transitions or detected component failures
- Conditions specified by various fault policies, some of which are user-configurable



Note

You can set fault thresholds on statistical measurements such as health scores, data traffic, or temperatures.

Procedure

Step 1 To retrieve the health score for a tenant named "3tierapp", send a REST query to the fabric such as the following:

Example:

```
https://apic-ip-address/api/node/mo/uni/tn-3tierapp.xml?query-target=self&rsp-subtreeinclude=health
```

Step 2 To retrieve statistics for a tenant named "3tierapp", send a REST query to the fabric such as the following:

Example:

```
https://apic-ip-address/api/node/mo/uni/tn-3tierapp.xml?query-target=self&rsp-subtreeinclude=stats
```

Step 3 To retrieve the faults for a leaf node, send a REST query to the fabric such as the following:

Example:

```
https://apic-ip-address/api/node/mo/topology/pod-1/node-103.xml?query-target=self&rsp-subtreeinclude=faults
```

Statistics

Configuring a Stats Monitoring Policy Using the REST API

To use statistics for monitoring and troubleshooting the fabric, you can configure a stats collection policy and a stats export policy to monitor many objects in the APIC.

Procedure

- Step 1** To create a stats collection policy using the REST API, send a POST request with XML such as the following:

Example:

```
<monEPGPol name="MonPol1" dn="uni/tn-tenant64/monepg-MonPol1">
    <monEPGTarget name="" descr="" scope="eventSevAsnP"/>
    <monEPGTarget name="" descr="" scope="faultSevAsnP"/>
    <monEPGTarget name="" descr="" scope="fvBD">
        <statsHierColl name="" descr="" histRet="inherited" granularity="5min"
adminState="inherited"/>
    </monEPGTarget>
    <monEPGTarget name="" descr="" scope="syslogRsDestGroup"/>
    <monEPGTarget name="" descr="" scope="syslogSrc"/>
    <monEPGTarget name="" descr="" scope="fvCtx"/>
    <statsHierColl name="" descr="" histRet="none" granularity="1w" adminState="enabled"/>
    <statsHierColl name="" descr="" histRet="none" granularity="1qtr" adminState="enabled"/>
    <statsHierColl name="" descr="" histRet="1w" granularity="1h" adminState="enabled"/>
    <statsHierColl name="" descr="" histRet="1d" granularity="15min" adminState="enabled"/>
    <statsHierColl name="" descr="" histRet="none" granularity="1year" adminState="enabled"/>
    <statsHierColl name="" descr="" histRet="none" granularity="1mo" adminState="enabled"/>
    <statsHierColl name="" descr="" histRet="1h" granularity="5min" adminState="enabled"/>
    <statsHierColl name="" descr="" histRet="10d" granularity="1d" adminState="enabled"/>
    <syslogSrc name="VRF64_SyslogSource" descr="" minSev="warnings" incl="faults">
        <syslogRsDestGroup tDn="uni/fabric/slgroup-tenant64_SyslogDest"/>
    </syslogSrc>
</monEPGPol>
```

- Step 2** To configure a stats export policy send a post with XML such as the following (you can use either JSON or XML format):

Example:

```
<statsExportP
    name="" descr="" frequency="stream" format="xml" compression="gzip">
    <statsDestP name="tenant64_statsExportDest" descr="" userName="" remotePort="0"
        remotePath="192.168.100.20" protocol="sftp" host="192.168.100.1">
            <fileRsARemoteHostToEpg tDn="uni/tn-mgmt/mgmtp-default/oob-default"/>
        </statsDestP>
</statsExportP>
```

Recovering a Disconnected Leaf

Recovering a Disconnected Leaf

If all fabric interfaces on a leaf are disabled (interfaces connecting a leaf to the spine) due to a configuration pushed to the leaf, connectivity to the leaf is lost forever and the leaf becomes inactive in the fabric. Trying to push a configuration to the leaf does not work because connectivity has been lost. This chapter describes how to recover a disconnected leaf.

Recovering a Disconnected Leaf Using the REST API

To recover a disconnected leaf, at least one of the fabric interfaces must be enabled using the following process. The remaining interfaces can be enabled using the GUI, REST API, or CLI.

To enable the first interface, post a policy using the REST API to delete the policy posted and bring the fabric ports Out-of-Service. You can post a policy to the leaf to bring the port that is Out-of-Service to In-Service as follows:



Note In the following examples, the assumption is that 1/49 is one of the leaf ports connecting to the spine.

Procedure

-
- Step 1** Clear the blacklist policy from the APIC (using the REST API).

Example:

```
$APIC_Address/api/policymgr/mo/.xml
<polUni>
    <fabricInst>
        <fabricOOServicePol>
            <fabricRsOosPath tDn="topology/pod-1/paths-$LEAF_Id/pathEp-[eth1/49]"
lc="blacklist" status ="deleted" />
        </fabricOOServicePol>
    </fabricInst>
</polUni>
```

- Step 2** Post a local task to the node itself to bring up the interfaces you want using **l1EthIfSetInServiceLTTask**.

Example:

```
$LEAF_Address/api/node/mo/topology/pod-1/node-$LEAF_Id/sys/action.xml
<actionLSubj oDn="sys/phys-[eth1/49]">
<l1EthIfSetInServiceLTTask adminSt='start' />
</actionLSubj>
```

Troubleshooting Contracts and Taboo Contracts with Permit and Deny Logging

Verifying Contracts, Taboo Contracts, and Filters Using the REST API

This topic provides the REST API XML to verify contracts, taboo contracts, and filters.

Procedure

- Step 1** Verify a contract for an EPG or an external network with XML such as the following example for a provider:

Example:

```
QUERY https://apic-ip-address/api/node/class/fvRsProv.xml
```

- Step 2** Verify a contract on an EPG with XML such as the following example for a consumer:

Example:

```
QUERY https://apic-ip-address/api/node/class/fvRsCons.xml
```

- Step 3** Verify exported contracts using XML such as the following example:

Example:

```
QUERY https://apic-ip-address/api/node/class/vzCPif.xml
```

- Step 4** Verify contracts for a VRF with XML such as the following example:

Example:

```
QUERY https://apic-ip-address/api/node/class/vzBrCP.xml
```

- Step 5** Verify taboo contracts with XML such as the following example:

Example:

```
QUERY https://apic-ip-address/api/node/class/vzTaboo.xml
```

For taboo contracts for an EPG, use the same query as for contracts for EPGs.

- Step 6** Verify filters using XML such as the following example:

Example:

```
QUERY https://apic-ip-address/api/node/class/vzFilter.xml
```

Viewing ACL Permit and Deny Logs Using the REST API

The following example shows how to view permit and deny log data for traffic flows, using the REST API:

Before You Begin

You must enable permit or deny logging, before you can view ACL contract permit and deny log data.

Procedure

Send the following query using the REST API:

```
GET
https://apic-ip-address/api/node/mo/uni/tn-sgladwin_t1.json?rsp-subtree-includes=stats&rsp-subtree-class=fvOverallHealthHist15min
{
  "totalCount": "1",
  "imdata": [
    {
      "fvTenant": {
        "attributes": {
          "childAction": "",
          "descr": "",
          "dn": "uni/tn-sgladwin_t1",
          "llcOwn": "local",
          "modTs": "2016-06-22T15:46:30.745+00:00",
          "monPolDn": "uni/tn-common/monepg-default",
          "name": "sgladwin_t1",
          "ownerKey": "",
          "ownerTag": "",
          "status": "",
          "uid": "15374"
        }
      }
    }
  ]
}
```

Troubleshooting Using Digital Optical Monitoring Statistics

Troubleshooting Using Digital Optical Monitoring With the REST API

To view DOM statistics using an XML REST API query:

Before You Begin

You must have previously enabled digital optical monitoring (DOM) on an interface, before you can view the DOM statistics for it.

Procedure

The following example shows how to view DOM statistics on a physical interface, eth1/25 on node-104, using a REST API query:

```
GET
https://apic-ip-address/api/node/mo/topology/pod-1/node-104/sys/phys-[eth1/25]/phys/domstats.xml?
query-target=children&target-subtree-class=ethpmDOMRxPwrStats&subscription=yes
```

The following response is returned:

```
response : {
  "totalCount": "1",
  "subscriptionId": "72057611234705430",
  "imdata": [
    {
      "ethpmDOMRxPwrStats": {
        "attributes": {
          "alert": "none",
          "childAction": "",
          "dn": "topology/pod-1/node-104/sys/phys[eth1/25]/phys/domstats/rxpower",
          "hiAlarm": "0.158490",
          "hiWarn": "0.079430",
          "lowPower": "0.000000",
          "lowPowerThreshold": "0.000000",
          "rxPower": "0.000000",
          "rxPowerThreshold": "0.000000",
          "txPower": "0.000000",
          "txPowerThreshold": "0.000000"
        }
      }
    }
  ]
}
```

```

    "loAlarm":"0.001050",
    "loWarn":"0.002630",
    "modTs":"never",
    "status":"",
    "value":"0.139170"}]}]}
  
```

Troubleshooting Using Port Tracking

Port Tracking Policy for Uplink Failure Detection

Uplink failure detection can be enabled in the fabric access global port tracking policy. The port tracking policy monitors the status of links between leaf switches and spine switches. When an enabled port tracking policy is triggered, the leaf switches take down all access interfaces on the switch that have EPGs deployed on them.


Note

In the advanced GUI, port tracking is located under **Fabric > Access Policies > Global Policies > Port Tracking**.

In the basic GUI, port tracking is located under **System > System Settings > Port Tracking**.

Depending on the model of leaf switch, each leaf switch can have 6, 8, or 12 uplink connections to each spine switch. The port tracking policy specifies the number of uplink connections that trigger the policy, and a delay timer for bringing the leaf switch access ports back up after the number of specified uplinks is exceeded.

The following example illustrates how a port tracking policy behaves:

- The leaf switches each have 6 active uplink connections to the spine switches.
- The port tracking policy specifies that the threshold of active uplink connections each leaf switch that triggers the policy is 2.
- The port tracking policy triggers when the number of active uplink connections from the leaf switch to the spine switches drops to 2.
- Each leaf switch monitors its uplink connections and triggers the port tracking policy according to the threshold specified in the policy.
- When the uplink connections come back up, the leaf switch waits for the delay timer to expire before bringing its access ports back up. This gives the fabric time to reconverge before allowing traffic to resume on leaf switch access ports. Large fabrics may need the delay timer to be set for a longer time.


Note

Use caution when configuring this policy. If the port tracking setting for the number of active spine links that triggers port tracking is too high, all leaf switch access ports will be brought down.

Port Tracking Using the REST API

Before You Begin

This procedure explains how to use the Port Tracking feature using the REST API.

Procedure

- Step 1** Turn on the Port Tracking feature using the REST API as follows (**admin state: on**):

```
<polUni>
<infraInfra dn="uni/infra">
<infraPortTrackPol name="default" delay="5" minlinks="4" adminSt="on">
</infraPortTrackPol>
</infraInfra>
</polUni>
```

- Step 2** Turn off the Port Tracking feature using the REST API as follows (**admin state: off**):

```
<polUni>
<infraInfra dn="uni/infra">
<infraPortTrackPol name="default" delay="5" minlinks="4" adminSt="off">
</infraPortTrackPol>
</infraInfra>
</polUni>
```

Removing Unwanted _ui_ Objects

Removing Unwanted _ui_ Objects Using the REST API

If you make changes with the Basic GUI or the NX-OS CLI before using the Advanced GUI, and objects appear in the Advanced GUI (with names prepended with `_ui_`), these objects can be removed by performing a REST API request to the API, containing the following:

- The Class name, for example `infraAccPortGrp`
- The Dn attribute, for example `dn="uni/infra/funcprof/accportgrp-_ui_l101_eth1--31"`
- The Status attribute set to `status="deleted"`

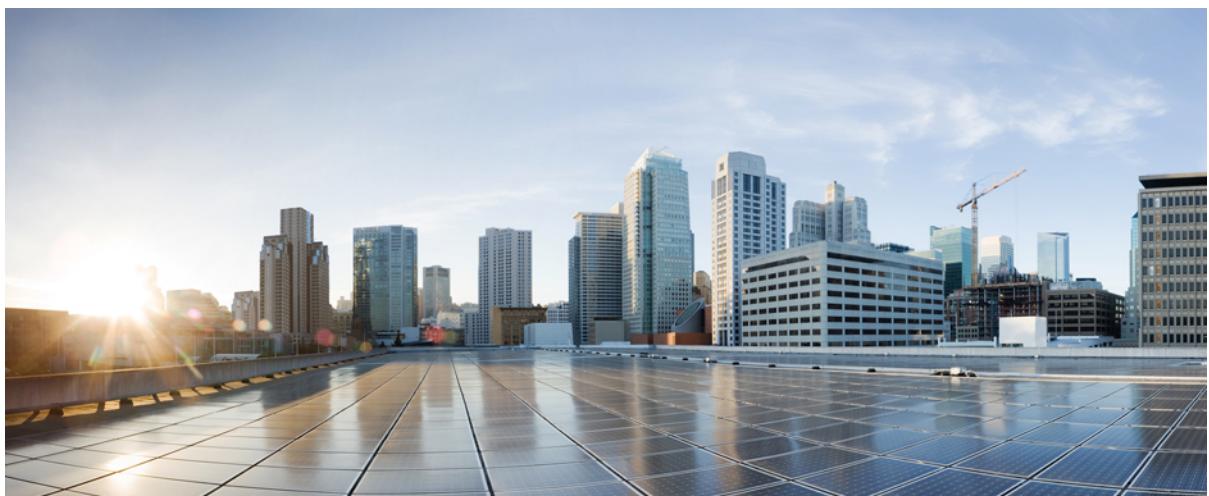
Perform the POST to the API with the following steps:

Procedure

- Step 1** Log on to a user account with write access to the object to be removed.

- Step 2** Send a POST to the API such as the following example:

```
POST https://192.168.20.123/api/mo/uni.xml
Payload:<infraAccPortGrp dn="uni/infra/funcprof/accportgrp-_ui_l101_eth1--31"
status="deleted"/>
```



PART ■ ■ ■

Part 3: Setting Up APIC and the Fabric Using the REST API

- [Managing APIC Clusters, page 113](#)
- [Configuring Tenant Policies, page 119](#)
- [Provisioning Core Services, page 157](#)
- [Provisioning Layer 2 Networks, page 177](#)
- [Provisioning Layer 3 Outside Connections, page 229](#)
- [Managing Layer 4 to Layer 7 Services, page 279](#)
- [Configuring QoS, page 313](#)
- [Configuring Security, page 319](#)



CHAPTER 9

Managing APIC Clusters

- [Cluster Management Guidelines, page 113](#)
- [Expanding and Contracting Clusters, page 114](#)
- [Managing Cluster High Availability, page 116](#)

Cluster Management Guidelines

Cluster Management Guidelines

The APIC cluster is comprised of multiple APIC controllers that provide operators a unified real time monitoring, diagnostic, and configuration management capability for the ACI fabric. To assure optimal system performance, follow the guidelines below for making changes to the APIC cluster.



Note

Prior to initiating a change to the cluster, always verify its health. When performing planned changes to the cluster, all controllers in the cluster should be healthy. If one or more of the APIC controllers' health status in the cluster is not "fully fit", remedy that situation before proceeding. Also, assure that cluster controllers added to the APIC are running the same version of firmware as the other controllers in the APIC cluster. See the *Cisco APIC Troubleshooting Guide* for more information on resolving APIC cluster health issues.

Follow these general guidelines when managing clusters:

- It is recommended that you have at least 3 active APICs in a cluster, and one or more standby APICs.
- Disregard cluster information from APICs that are not currently in the cluster; they do not provide accurate cluster information.
- Cluster slots contain an APIC `ChassisID`. Once you configure a slot, it remains unavailable until you decommission the APIC with the assigned `ChassisID`.
- If an APIC firmware upgrade is in progress, wait for it to complete and the cluster to be fully fit before proceeding with any other changes to the cluster.
- Always decommission an APIC cluster controller before doing a power cycle, then add it back to the cluster. Failure to follow this guideline can corrupt the APIC cluster database shards residing on the

cluster, which will require doing a wipe of the controller, then a cluster synchronization to restore a valid copy of the APIC cluster database from the other APIC controllers in the cluster.

- When an APIC cluster is split into two or more groups, the ID of a node is changed and the changes are not synchronized across all APICs. This can cause inconsistency in the node IDs between APICs and also the affected leaf nodes may not appear in the inventory in the APIC GUI. When you split an APIC cluster, decommission the affected leaf nodes from APIC and register them again, so that the inconsistency in the node IDs is resolved and the health status of the APICs in a cluster are in a fully fit state

This section contains the following topics:

Expanding and Contracting Clusters

Expanding the APIC Cluster Size

Follow these guidelines to expand the APIC cluster size:

- Schedule the cluster expansion at a time when the demands of the fabric workload will not be impacted by the cluster expansion.
- If one or more of the APIC controllers' health status in the cluster is not "fully fit", remedy that situation before proceeding.
- Stage the new APIC controller(s) according to the instructions in their hardware installation guide. Verify in-band connectivity with a PING test.
- Increase the cluster target size to be equal to the existing cluster size controller count plus the new controller count. For example, if the existing cluster size controller count is 3 and you are adding 3 controllers, set the new cluster target size to 6. The cluster proceeds to sequentially increase its size one controller at a time until all new controllers are included in the cluster.



Note Cluster expansion stops if an existing APIC controller becomes unavailable. Resolve this issue before attempting to proceed with the cluster expansion.

- Depending on the amount of data the APIC must synchronize upon the addition of each appliance, the time required to complete the expansion could be more than 10 minutes per appliance. Upon successful expansion of the cluster, the APIC operational size and the target size will be equal.



Note Allow the APIC to complete the cluster expansion before making additional changes to the cluster.

Expanding the Cisco APIC Cluster

Expanding the Cisco APIC cluster is the operation to increase any size mismatches, from a cluster size of N to size N+1, within legal boundaries. The operator sets the administrative cluster size and connects the APICs with the appropriate cluster IDs, and the cluster performs the expansion.

During cluster expansion, regardless of in which order you physically connect the APIC controllers, the discovery and expansion takes place sequentially based on the APIC ID numbers. For example, APIC2 is discovered after APIC1, and APIC3 is discovered after APIC2 and so on until you add all the desired APICs to the cluster. As each sequential APIC is discovered, a single data path or multiple data paths are established, and all the switches along the path join the fabric. The expansion process continues until the operational cluster size reaches the equivalent of the administrative cluster size.

Expanding the APIC Cluster Using the REST API

The cluster drives its actual size to the target size. If the target size is higher than the actual size, the cluster size expands.

Procedure

- Step 1** Set the target cluster size to expand the APIC cluster size.

Example:

```
POST
https://<IP address>/api/node/mo/uni/controller.xml
<infraClusterPol name='default' size=3/>
```

- Step 2** Physically connect the APIC controllers that you want to add to the cluster.
-

Contracting the Cisco APIC Cluster

Contracting the Cisco APIC cluster is the operation to decrease any size mismatches, from a cluster size of N to size N -1, within legal boundaries. As the contraction results in increased computational and memory load for the remaining APICs in the cluster, the decommissioned APIC cluster slot becomes unavailable by operator input only.

During cluster contraction, you must begin decommissioning the last APIC in the cluster first and work your way sequentially in reverse order. For example, APIC4 must be decommissioned before APIC3, and APIC3 must be decommissioned before APIC2.

Contracting the APIC Cluster Using the REST API

The cluster drives its actual size to the target size. If the target size is lower than the actual size, the cluster size contracts.

Procedure

- Step 1** Set the target cluster size so as to contract the APIC cluster size.

Example:

```
POST
https://<IP address>/api/node/mo/uni/controller.xml
<infraClusterPol name='default' size=1/>
```

- Step 2** Decommission APIC3 on APIC1 for cluster contraction.

Example:

```
POST
https://<IP address>/api/node/mo/topology/pod-1/node-1/av.xml
<infraWiNode id=3 adminSt='out-of-service'/'>
```

Step 3 Decommission APIC2 on APIC1 for cluster contraction.

Example:

```
POST
https://<IP address>/api/node/mo/topology/pod-1/node-1/av.xml
<infraWiNode id=2 adminSt='out-of-service'/'>
```

Managing Cluster High Availability

About High Availability for APIC Cluster

The High Availability functionality for an APIC cluster enables you to operate the APICs in a cluster in an Active/Standby mode. In an APIC cluster, the designated active APICs share the load and the designated standby APICs can act as a replacement for any of the APICs in an active cluster.

An admin user can set up the High Availability functionality when the APIC is launched for the first time. See *Cisco APIC Getting Started Guide*. It is recommended that you have at least 3 active APICs in a cluster, and one or more standby APICs. An admin user will have to initiate the switch over to replace an active APIC with a standby APIC.

Important Notes

- The standby APIC will be automatically updated with firmware updates to keep the backup APIC at same firmware version as the active cluster.
- During an upgrade process, once all the active APICs are upgraded, the standby APIC will also be upgraded automatically.
- Temporary IDs are assigned to standby APICs. After a standby APIC is switched over to a active APIC, a new ID is assigned.
- Admin login is not enabled on standby APIC. To troubleshoot HA, you must log in to the standby using SSH as *rescue-user*.
- During switch over the replaced active APIC will be powered down, to prevent connectivity to the replaced APIC.
- Switch over will fail under the following conditions:
 - If there is no connectivity to the standby APIC.
 - If the firmware version of the standby APIC is not the same as that of the active cluster.
- After switching over a standby APIC to active, if it was the only standby, you must configure a new standby.
- The following limitations are observed for retaining out of band address for standby APIC after a fail over.

- Standby(new active) APIC may not retain its out of band address if more than 1 active APICs are down or unavailable.
- Standby(new active) APIC may not retain its out of band address if it is in a different subnet than active APIC.
- Standby(new active) APIC may not retain its IPv6 out of band address.

**Note**

In case you observe any of the limitations, in order to retain standby APICs out of band address, you must manually change the OOB policy for replaced APIC after the replace operation is completed successfully.

- It is recommended to keep standby APICs in same POD as the active APICs it may replace.

Switching Over Active APIC with Standby APIC Using REST API

Use this procedure to switch over an active APIC with standby APIC using REST API.

Procedure

Switch over active APIC with standby APIC.

```
URL for POST: https://ip
address/api/node/mo/topology/pod-initiator_pod_id/node-initiator_id/av.xml
Body: <infraWiNode id=outgoing_apic_id targetMbSn=backup-serial-number/>
where initiator_id = id of an active APIC other than the APIC being replaced.
pod-initiator_pod_id = pod ID of the active APIC
backup-serial-number = serial number of standby APIC
```

Example:

```
https://ip address/api/node/mo/topology/pod-1/node-1/av.xml
<infraWiNode id=2 targetMbSn=FCH1750V00Q/>
```




Configuring Tenant Policies

- Basic Tenant Configuration, page 119
- Tenants in Multiple Private Networks, page 120
- Tenant Policy Example, page 123
- EPGs, page 133
- Intra-EPG Isolation, page 136
- Microsegmentation, page 141
- Application Profiles, page 144
- Contracts, Taboo Contracts, and Preferred Groups, page 148

Basic Tenant Configuration

Creating a Tenant, VRF, and Bridge Domain Using the REST API

Procedure

Step 1 Create a tenant.

Example:

```
POST https://apic-ip-address/api/mo/uni.xml  
<fvTenant name="ExampleCorp"/>
```

When the POST succeeds, you see the object that you created in the output.

Step 2 Create a VRF and bridge domain.

Note The Gateway Address can be an IPv4 or an IPv6 address. For more about details IPv6 gateway address, see the related KB article, *KB: Creating a Tenant, VRF, and Bridge Domain with IPv6 Neighbor Discovery*.

Example:

```
URL for POST: https://apic-ip-address/api/mo/uni/tn-ExampleCorp.xml
```

```
<fvTenant name="ExampleCorp">
  <fvCtx name="pvn1"/>
  <fvBD name="bd1">
    <fvRsCtx tnFvCtxName="pvn1"/>
    <fvSubnet ip="10.10.100.1/24"/>
  </fvBD>
</fvTenant>
```

Note If you have a public subnet when you configure the routed outside, you must associate the bridge domain with the outside configuration.

Tenants in Multiple Private Networks

About Multiple Private Networks with Inter-Tenant Communication

- This use case may be typical for environments where an ACI administrator wishes to create multiple tenants with the ability to support inter-tenant communications.

This method has the following advantages and disadvantages:

Advantages:

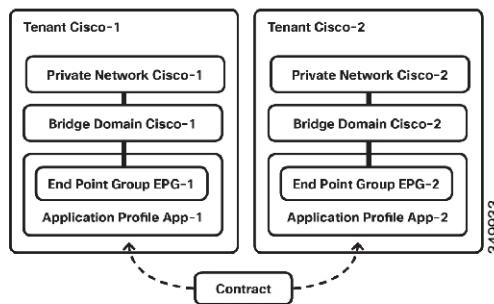
- Each tenant container can be managed separately
- Allows for maximum isolation between tenants

Disadvantages:

- Tenant address space must be unique

From a containment and relationship perspective, this topology looks as follows:

Figure 6: Multiple Private Networks with Inter-Tenant Communication



Configuring Multiple Private Networks with Inter-Tenant Communication Using the REST API

Configure the Cisco-1 and Cisco-2 private networks, with communication between them, using the REST API in the following steps:

Procedure

- Step 1** Configure Cisco-1 tenant using the following XML posted to the APIC REST API:

Example:

```
<fvTenant dn="uni/tn-Cisco1" name="Cisco1">

<vzBrCP name="ICMP" scope="global">
<vzSubj consMatchT="AtleastOne" name="icmp" provMatchT="AtleastOne"
revFltPorts="yes">
<vzRsSubjFiltAtt tnVzFilterName="icmp"/>
</vzSubj>
</vzBrCP>

<vzCPIf dn="uni/tn-Cisco1/cif-ICMP" name="ICMP">

<vzRsIf consMatchT="AtleastOne" name="icmp" provMatchT="AtleastOne"
revFltPorts="yes">
<vzRsSubjFiltAtt tDn="uni/tn-Cisco2/brc-default"/>
</vzRsIf>
</vzCPIf>
<fvCtx knwMcastAct="permit" name="CiscoCtx" pcEnfPref="enforced"/>

<fvBD arpFlood="yes" mac="00:22:BD:F8:19:FF" name="CiscoBD2" unicastRoute="yes"
unkMacUcastAct="flood" unkMcastAct="flood">
<fvRsCtx tnFvCtxName="CiscoCtx2"/>
</fvBD>
<fvBD arpFlood="yes" name="CiscoBD" unicastRoute="yes" unkMacUcastAct="flood"
unkMcastAct="flood">
<fvRsCtx tnFvCtxName="CiscoCtx"/>
</fvBD>
<fvAp name="CCO">
<fvAEPg matchT="AtleastOne" name="EPG1">
<fvRsPathAtt encap="vlan-1202" instrImedcy="immediate" mode="native"
tDn="topology/pod-1/paths-202/pathep-[eth1/2]"/>
<fvSubnet ip="172.16.1.1/24" scope="private,shared"/>

<fvRsDomAtt instrImedcy="lazy" resImedcy="lazy" tDn="uni/phys-
PhysDomainforCisco"/>

<fvRsBd tnFvBDName="CiscoBD"/>
<fvRsProv matchT="AtleastOne" tnVzBrCPName="ICMP"/>
</fvAEPg>
</fvAp>
</fvTenant>

<fvRsBd tnFvBDName="CiscoBD"/>
<fvRsProv matchT="AtleastOne" tnVzBrCPName="ICMP"/>
</fvAEPg>
</fvAp>
</fvTenant>
```

- Step 2** Configure Cisco-2 tenant using the following XML posted to the APIC REST API:

Example:

```
<fvTenant dn="uni/tn-Cisco2" name="Cisco2">
<fvCtx knwMcastAct="permit" name="CiscoCtx" pcEnfPref="enforced"/>
<fvBD arpFlood="yes" mac="00:22:BD:F8:19:FF" name="CiscoBD2" unicastRoute="yes"
unkMacUcastAct="flood" unkMcastAct="flood">
<fvRsCtx tnFvCtxName="CiscoCtx"/>
</fvBD>
<fvBD arpFlood="yes" name="CiscoBD" unicastRoute="yes" unkMacUcastAct="flood"
unkMcastAct="flood">
<fvRsCtx tnFvCtxName="CiscoCtx"/>
</fvBD>
```

```

<fvAp name="CCO">
<fvAEPg matchT="AtleastOne" name="EPG2">
<fvRsPathAtt encaps="vlan-1202" instrImedcy="immediate" mode="native"
tDn="topology/pod-1/paths-201/pathep-[eth1/2]"/>
<fvSubnet ip="172.16.1.1/24" scope="private,shared"/>

<fvRsDomAtt instrImedcy="lazy" resImedcy="lazy" tDn="uni/phys-
PhysDomainforCisco"/>
<fvRsBd tnFvBDName="CiscoBD"/>
<fvRsConsIf matchT="AtleastOne" tnVzBrCPIfName="ICMP"/>
</fvAEPg>
</fvAp>
</fvTenant>

```

About Multiple Private Networks with Intra-Tenant Communication

Another use case that may be desirable to support is the option to have a single tenant with multiple private networks. This may be a result of needing to provide multitenancy at a network level, but not at a management level. It may also be caused by needing to support overlapping subnets within a single tenant, due to mergers and acquisitions or other business changes.

This method has the following advantages and disadvantages:

Advantages:

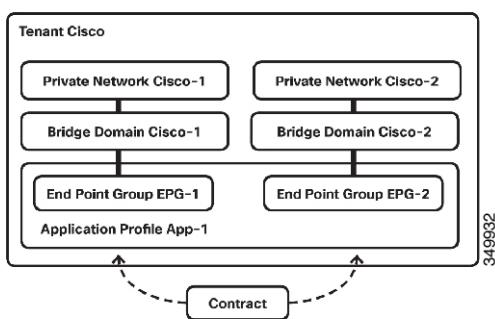
- Ability to have overlapping subnets within a single tenant

Disadvantages:

- EPGs residing in overlapping subnets cannot have policy applied between one another

The object containment for this particular setup can be depicted as shown below:

Figure 7: Multiple Private Networks with Intra-Tenant Communication



Configuring Multiple Tenants with Intra-Tenant Communication Using the REST API

Procedure

Configure the Tenant Cisco, with Cisco-1 and Cisco-2 networks, using the following XML posted to the APIC REST API:

Example:

```

<fvTenant dn="uni/tn-Cisco" name="Cisco">
<vzBrCP name="ICMP" scope="tenant">
<vzSubj consMatchT="AtleastOne" name="icmp" provMatchT="AtleastOne"
revFltPorts="yes">
<vzRsSubjFiltAtt tnVzFilterName="icmp"/>
</vzSubj>
</vzBrCP>
<fvCtx knwMcastAct="permit" name="CiscoCtx" pcEnfPref="enforced"/>
<fvCtx knwMcastAct="permit" name="CiscoCtx2" pcEnfPref="enforced"/>
<fvBD arpFlood="yes" mac="00:22:BD:F8:19:FF" name="CiscoBD2" unicastRoute="yes"
unkMacUcastAct="flood" unkMcastAct="flood">
<fvRsCtx tnFvCtxName="CiscoCtx2"/>
</fvBD>
<fvBD arpFlood="yes" mac="00:22:BD:F8:19:FF" name="CiscoBD" unicastRoute="yes"
unkMacUcastAct="flood" unkMcastAct="flood">
<fvRsCtx tnFvCtxName="CiscoCtx"/>
</fvBD>
<fvAp name="CCO">
<fvAEPg matchT="AtleastOne" name="Web">
<fvRsCons tnVzBrCPName="ICMP"/>
<fvRsPathAtt encaps="vlan-1201" instrImedcy="immediate" mode="native"
tDn="topology/pod-1/paths-201/pathep-[eth1/16]"/>
<fvSubnet ip="172.16.2.1/24" scope="private,shared"/>
<fvRsDomAtt instrImedcy="lazy" resImedcy="lazy" tDn="uni/phys-
PhysDomainforCisco"/>
<fvRsBd tnFvBDName="CiscoBD2"/>
</fvAEPg>
<fvAEPg matchT="AtleastOne" name="App">
<fvRsPathAtt encaps="vlan-1202" instrImedcy="immediate" mode="native"
tDn="topology/pod-1/paths-202/pathep-[eth1/2]"/>
<fvSubnet ip="172.16.1.1/24" scope="private,shared"/>
<fvRsDomAtt instrImedcy="lazy" resImedcy="lazy" tDn="uni/phys-
PhysDomainforCisco"/>
<fvRsBd tnFvBDName="CiscoBD"/>
<fvRsProv matchT="AtleastOne" tnVzBrCPName="ICMP"/>
</fvAEPg>
</fvAp>
</fvTenant>

```

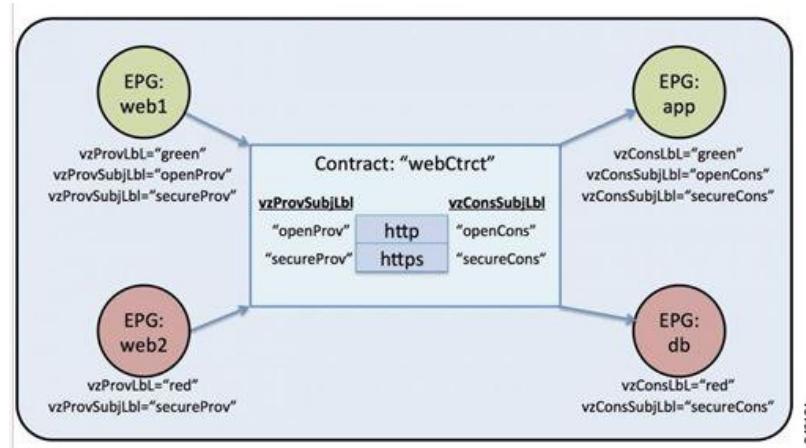
Tenant Policy Example

Tenant Policy Example Overview

The description of the tenant policy example in this appendix uses XML terminology (http://en.wikipedia.org/wiki/XML#Key_terminology). This example demonstrates how basic APIC policy

model constructs are rendered into the XML code. The following figure provides an overview of the tenant policy example.

Figure 8: EPGs and Contract Contained in Tenant Solar



In the figure, according to the contract called `webCtrct` and the EPG labels, the green-labeled EPG:`web1` can communicate with green-labeled EPG:`app` using both `http` and `https`, the red-labeled EPG:`web2` can communicate with the red-labeled EPG:`db` using only `https`.

Tenant Policy Example XML Code

```

<polUni>
    <fvTenant name="solar">

        <vzFilter name="Http">
            <vzEntry name="e1">
                etherT="ipv4"
                prot="tcp"
                dFromPort="80"
                dToPort="80"/>
            </vzEntry>
        </vzFilter>

        <vzFilter name="Https">
            <vzEntry name="e1">
                etherT="ipv4"
                prot="tcp"
                dFromPort="443"
                dToPort="443"/>
            </vzEntry>
        </vzFilter>

        <vzBrCP name="webCtrct">
            <vzSubj name="http" revFltPorts="true" provmatchT="All">
                <vzRsSubjFiltAtt tnVzFilterName="Http"/>
                <vzRsSubjGraphAtt graphName="G1" termNodeName="TProv"/>
                <vzProvSubjLbl name="openProv"/>
                <vzConsSubjLbl name="openCons"/>
            </vzSubj>
            <vzSubj name="https" revFltPorts="true" provmatchT="All">
                <vzProvSubjLbl name="secureProv"/>
                <vzConsSubjLbl name="secureCons"/>
                <vzRsSubjFiltAtt tnVzFilterName="Https"/>
                <vzRsOutTermGraphAtt graphName="G2" termNodeName="TProv"/>
            </vzSubj>
        </vzBrCP>
    </fvTenant>
</polUni>

```

```

<fvCtx name="solarctx1"/>

<fvBD name="solarBD1">
    <fvRsCtx tnFvCtxName="solarctx1" />
    <fvSubnet ip="11.22.22.20/24">
        <fvRsBDsubnetToProfile
            tnL3extOutName="rout1"
            tnRtctrlProfileName="profExport"/>
    </fvSubnet>
    <fvSubnet ip="11.22.22.211/24">
        <fvRsBDsubnetToProfile
            tnL3extOutName="rout1"
            tnRtctrlProfileName="profExport"/>
    </fvSubnet>
</fvBD>

<fvAp name="sap">
    <fvAEPg name="web1">
        <fvRsBd tnFvBDName="solarBD1" />
        <fvRsDomAtt tDn="uni/vmmp-VMware/dom-mininet" />
        <fvRsProv tnVzBrCPName="webCtrct" matchT="All">
            <vzProvSubjLbl name="openProv"/>
            <vzProvSubjLbl name="secureProv"/>
            <vzProvLbl name="green"/>
        </fvRsProv>
    </fvAEPg>
    <fvAEPg name="web2">
        <fvRsBd tnFvBDName="solarBD1" />
        <fvRsDomAtt tDn="uni/vmmp-VMware/dom-mininet" />
        <fvRsProv tnVzBrCPName="webCtrct" matchT="All">
            <vzProvSubjLbl name="secureProv"/>
            <vzProvLbl name="red"/>
        </fvRsProv>
    </fvAEPg>
    <fvAEPg name="app">
        <fvRsBd tnFvBDName="solarBD1" />
        <fvRsDomAtt tDn="uni/vmmp-VMware/dom-mininet" />
        <fvRsCons tnVzBrCPName="webCtrct">
            <vzConsSubjLbl name="openCons"/>
            <vzConsSubjLbl name="secureCons"/>
            <vzConsLbl name="green"/>
        </fvRsCons>
    </fvAEPg>
    <fvAEPg name="db">
        <fvRsBd tnFvBDName="solarBD1" />
        <fvRsDomAtt tDn="uni/vmmp-VMware/dom-mininet" />
        <fvRsCons tnVzBrCPName="webCtrct">
            <vzConsSubjLbl name="secureCons"/>
            <vzConsLbl name="red"/>
        </fvRsCons>
    </fvAEPg>
</fvAp>
</fvTenant>
</polUni>

```

Tenant Policy Example Explanation

This section contains a detailed explanation of the tenant policy example.

Policy Universe

The policy universe contains all the tenant-managed objects where the policy for each tenant is defined.

```
<polUni>
```

This starting tag, <polUni>, in the first line indicates the beginning of the policy universe element. This tag is matched with </polUni> at the end of the policy. Everything in between is the policy definition.

Tenant Policy Example

The <fvTenant> tag identifies the beginning of the tenant element.

```
<fvTenant name="solar">
```

All of the policies for this tenant are defined in this element. The name of the tenant in this example is solar. The tenant name must be unique in the system. The primary elements that the tenant contains are filters, contracts, outside networks, bridge domains, and application profiles that contain EPGs.

Filters

The filter element starts with a <vzFilter> tag and contains elements that are indicated with a <vzEntry> tag.

The following example defines "Http" and "Https" filters. The first attribute of the filter is its name and the value of the name attribute is a string that is unique to the tenant. These names can be reused in different tenants. These filters are used in the subject elements within contracts later on in the example.

```
<vzFilter name="Http">
    <vzEntry name="e1" etherT="ipv4" prot="tcp" dFromPort="80" dToPort="80"/>
</vzFilter>

<vzFilter name="Https">
    <vzEntry name="e1" etherT="ipv4" prot="tcp" dFromPort="443" dToPort="443"/>
</vzFilter>
```

This example defines these two filters: Http and Https. The first attribute of the filter is its name and the value of the name attribute is a string that is unique to the tenant, i.e. these names can be reused in different tenants. These filters will be used in the subject elements within contracts later on in the example.

Each filter can have one or more entries where each entry describes a set of Layer 4 TCP or UDP port numbers. Some of the possible attributes of the <vzEntry> element are as follows:

- name
- prot
- dFromPort
- dToPort
- sFromPort
- sToPort
- etherT
- ipFlags
- arpOpc
- tcpRules

In this example, each entry's name attribute is specified. The name is an ASCII string that must be unique within the filter but can be reused in other filters. Because this example does not refer to a specific entry later on, it is given a simple name of "e1".

The EtherType attribute, `etherT`, is next. It is assigned the value of `ipv4` to specify that the filter is for IPv4 packets. There are many other possible values for this attribute. Common ones include `ARP`, `RARP`, and `IPv6`. The default is `unspecified` so it is important to assign it a value.

Following the EtherType attribute is the `prot` attribute that is set to `tcp` to indicate that this filter is for TCP traffic. Alternate protocol attributes include `udp`, `icmp`, and `unspecified` (default).

After the protocol, the destination TCP port number is assigned to be in the range from 80 to 80 (exactly TCP port 80) with the `dFromPort` and `dToPort` attributes. If the from and to are different, they specify a range of port numbers.

In this example, these destination port numbers are specified with the attributes `dFromPort` and `dToPort`. However, when they are used in the contract, they should be used for the destination port from the TCP client to the server and as the source port for the return traffic. See the attribute `revFltPorts` later in this example for more information.

The second filter does essentially the same thing, but for port 443 instead.

Filters are referred to by subjects within contracts by their target distinguished name, `tDn`. The `tDn` name is constructed as follows:

```
uni/tn-<tenant name>/flt-<filter name>
```

For example, the `tDn` of the first filter above is `uni/tn-coke/flt-Http`. The second filter has the name `uni/tn-coke/flt-Https`. In both cases, `solar` comes from the tenant name.

Contracts

The contract element is tagged `vzBrCP` and it has a `name` attribute.

```
<vzBrCP name="webCtrct">
  <vzSubj name="http" revFltPorts="true" provmatchT="All">
    <vzRsSubjFiltAtt tnVzFilterName="Http"/>
    <vzRsSubjGraphAtt graphName="G1" termNodeName="TProv"/>
    <vzProvSubjLbl name="openProv"/>
    <vzConsSubjLbl name="openCons"/>
  </vzSubj>
  <vzSubj name="https" revFltPorts="true" provmatchT="All">
    <vzProvSubjLbl name="secureProv"/>
    <vzConsSubjLbl name="secureCons"/>
    <vzRsFiltAtt tnVzFilterName="Https "/>
    <vzRsOutTermGraphAtt graphName="G2" termNodeName="TProv"/>
  </vzSubj>
</vzBrCP>
```

Contracts are the policy elements between EPGs. They contain all of the filters that are applied between EPGs that produce and consume the contract. The contract element is tagged `vzBrCP` and it has a `name` attribute. Refer to the object model reference documentation for other attributes that can be used in the contract element. This example has one contract named `webCtrct`.

The contract contains multiple subject elements where each subject contains a set of filters. In this example, the two subjects are `http` and `https`.

The contract is later referenced by EPGs that either provide or consume it. They reference it by its name in the following manner:

```
uni/tn-[tenant-name]/brc-[contract-name]
```

`tenant-name` is the name of the tenant, “`solar`” in this example, and the `contract-name` is the name of the contract. For this example, the `tDn` name of the contract is `uni/tn-solar/brc-webCtrct`.

Subjects

The subject element starts with the tag `vzSubj` and has three attributes: `name`, `revFltPorts`, and `matchT`. The `name` is simply the ASCII name of the subject.

`revFltPorts` is a flag that indicates that the Layer 4 source and destination ports in the filters of this subject should be used as specified in the filter description in the forward direction (that is, in the direction of from consumer to producer EPG), and should be used in the opposite manner for the reverse direction. In this example, the “http” subject contains the “Http” filter that defined TCP destination port 80 and did not specify the source port. Because the `revFltPorts` flag is set to true, the policy will be TCP **destination port 80** and any source port for traffic from the consumer to the producer, and it will be TCP destination port any and **source port 80** for traffic from the producer to the consumer. The assumption is that the consumer initiates the TCP connection to the producer (the consumer is the client and the producer is the server).

The default value for the `revFltPrt`s attribute is `false` if it is not specified.

Labels

The match type attribute, `provmatchT` (for provider matching) and `consmatchT` (for consumer matching) determines how the subject labels are compared to determine if the subject applies for a given pair of consumers and producers. The following match type values are available:

- All
- AtLeastOne (default)
- None
- ExactlyOne

When deciding whether a subject applies to the traffic between a producer and consumer EPG, the match attribute determines how the subject labels that are defined (or not) in those EPGs should be compared to the labels in the subject. If the match attribute value is set to `All`, it only applies to the providers whose provider subject labels, `vzProvSubjLbl`, match all of the `vzProvSubjLbl` labels that are defined in the subject. If two labels are defined, both must also be in the provider. If a provider EPG has 10 labels, as long as all of the provider labels in the subject are present, a match is confirmed. A similar criteria is used for the consumers that use the `vzConsSubjLbl`. If the `matchT` attribute value is `AtLeastOne`, only one of the labels must match. If the `matchT` attribute is `None`, the match only occurs if none of the provider labels in the subject match the provider labels of the provider EPGs and similarly for the consumer.

If the producer or consumer EPGs do not have any subject labels and the subject does not have any labels, a match occurs for `All`, `AtLeastOne`, and `None` (if you do not use labels, the subject is used and the `matchT` attribute does not matter).

An optional attribute of the subject not shown in the example is `prio` where the priority of the traffic that matches the filter is specified. Possible values are `gold`, `silver`, `bronze`, or `unspecified` (default).

In the example, the subject element contains references to filter elements, subject label elements, and graph elements. `<vzRsSubjFiltAtt tDn="uni/tn-coke/flt-Http"/>` is a reference to a previously defined filter. This element is identified by the `vzRsSubjFiltAtt` tag.

`<vzRsSubjGraphAtt graphName="G1" termNodeName="TProv"/>` defines a terminal connection.

`<vzProvSubjLbl name="openProv"/>` defines a provider label named “openProv”. The label is used to qualify or filter which subjects get applied to which EPGs. This particular one is a provider label and the corresponding

consumer label is identified by the tag `vzConsSubjLbl`. These labels are matched with the corresponding label of the provider or consumer EPG that is associated with the current contract. If a match occurs according to the `matchT` criteria described above, a particular subject applies to the EPG. If no match occurs, the subject is ignored.

Multiple provider and consumer subject labels can be added to a subject to allow more complicated matching criteria. In this example, there is just one label of each type on each subject. However, the labels on the first subject are different from the labels on the second subject, which allows these two subjects to be handled differently depending on the labels of the corresponding EPGs. The order of the elements within the subject element does not matter.

VRF

The Virtual Routing and Forwarding (VRF) (also known as a context or private network) is identified by the `fvCtx` tag and contains a name attribute.

A tenant can contain multiple VRFs. For this example, the tenant uses one VRF named “solartx1”. The name must be unique within the tenant.

The VRF defines a Layer 3 address domain. All of the endpoints within the Layer 3 domain must have unique IPv4 or IPv6 addresses, because it is possible to directly forward packets between these devices if the policy allows it.

Although a VRF defines a unique IP address space, the corresponding subnets are defined within bridge domains. Each bridge domain is then associated with the VRF.

Bridge Domains

The bridge domain element is identified with the `fvBD` tag and has a name attribute.

```
<fvBD name="solarBD1">
    <fvRsCtx tnFvCtxName="solarctx1" />
    <fvSubnet ip="11.22.22.20/24">
        <fvRsBDSubnetToProfile
            tnL3extOutName="rout1"
            tnRtctrlProfileName="profExport" />
    </fvSubnet>
    <fvSubnet ip="11.22.23.211/24">
        <fvRsBDSubnetToProfile
            tnL3extOutName="rout1"
            tnRtctrlProfileName="profExport" />
    </fvSubnet>
</fvBD>
```

Within the bridge domain element, subnets are defined and a reference is made to the corresponding Virtual Routing and Forwarding (VRF) instance (also known as a context or private network). Each bridge domain must be linked to a VRF and have at least one subnet.

This example uses one bridge domain named “solarBD1”. In this example, the “solarctx1” VRF is referenced by using the element tagged `fvRsCtx` and the `tnFvCtxName` attribute is given the value “solarctx1”. This name comes from the VRF defined above.

The subnets are contained within the bridge domain and a bridge domain can contain multiple subnets. This example defines two subnets. All of the addresses used within a bridge domain must fall into one of the address ranges that is defined by the subnets. However, the subnet can also be a supernet which is a very large subnet that includes many addresses that might never be used. Specifying one giant subnet that covers all current future addresses can simplify the bridge domain specification. However, different subnets must not overlap.

within a bridge domain or with subnets defined in other bridge domains that are associated with the same VRF. Subnets can overlap with other subnets that are associated with other VRFs.

The subnets described above are 11.22.22.xx/24 and 11.22.23.xx/24. However, the full 32 bits of the address is given even though the mask says that only 24 are used, because this IP attribute also identifies the full IP address for the router in that subnet. In the first case, the router IP address (default gateway) is 11.22.22.20 and for the second subnet, it is 11.22.23.211.

The entry 11.22.22.20/24 is equivalent to the following, but in compact form:

- Subnet: 11.22.22.00
- Subnet Mask: 255.255.255.0
- Default gateway: 11.22.22.20

Application Profiles

The start of the application profile is indicated by the `fvAp` tag and has a name attribute.

```
<fvAp name="sap">
```

This example has one application network profile and it is named “sap.”

The application profile is a container that holds the EPGs. EPGs can communicate with other EPGs in the same application profile and with EPGs in other application profiles. The application profile is simply a convenient container that is used to hold multiple EPGs that are logically related to one another. They can be organized by the application they provide such as “sap,” by the function they provide such as “infrastructure,” by where they are in the structure of the data center such as “DMZ,” or whatever organizing principle the administrator chooses to use.

The primary object that the application profile contains is an endpoint group (EPG). In this example, the “sap” application profile contains 4 EPGs: web1, web2, app, and db.

Endpoints and Endpoint Groups (EPGs)

EPGs begin with the tag `fvAEPg` and have a name attribute.

```
<fvAEPg name="web1">
  <fvRsBd tnFvBDName="solarBD1" />
  <fvRsDomAtt tDn="uni/vmmp-VMware/dom-mininet" />
  <fvRsProv tnVzBrCPName="webCtrct" matchT ="All">
    <vzProvSubjLbl name="openProv"/>
    <vzProvSubjLbl name="secureProv"/>
    <vzProvLbl name="green"/>
  </fvRsProv>
</fvAEPg>
```

The EPG is the most important fundamental object in the policy model. It represents a collection of endpoints that are treated in the same fashion from a policy perspective. Rather than configure and manage those endpoints individually, they are placed within an EPG and are managed as a collection or group.

The EPG object is where labels are defined that govern what policies are applied and which other EPGs can communicate with this EPG. It also contains a reference to the bridge domain that the endpoints within the EPG are associated with as well as which virtual machine manager (VMM) domain they are associated with. VMM allows virtual machine mobility between two VM servers instantaneously with no application downtime.

The first EPG in the example is named “web1.” The `fvRsBd` element within the EPG defines which bridge domain that it is associated with. The bridge domain is identified by the value of the `tnFvBDName` attribute.

This EPG is associated with the “solarBD1” bridge domain named in the “Bridge Domain” section above. The binding to the bridge domain is used by the system to understand what the default gateway address should be for the endpoints in this EPG. It does not imply that the endpoints are all in the same subnet or that they can only communicate through bridging. Whether an endpoint’s packets are bridged or routed is determined by whether the source endpoint sends the packet to its default gateway or the final destination desired. If it sends the packet to the default gateway, the packet is routed.

The VMM domain used by this EPG is identified by the `fvRsDomAtt` tag. This element references the VMM domain object defined elsewhere. The VMM domain object is identified by its `tDn` name attribute. This example shows only one VMM domain called “uni/vmmp-VMware/dom-mininet.”

The next element in the “web1” EPG defines which contract this EPG provides and is identified by the `fvRsProv` tag. If “web1” were to provide multiple contracts, there would be multiple `fvRsProv` elements. Similarly, if it were to consume one or more contracts, there would be `fvRsCons` elements as well.

The `fvRsProv` element has a required attribute that is the name of the contract that is being provided. “web1” is providing the contract “webCtrct” that was defined earlier that was called `tDn=“uni/tn-coke/brc-webCtrct”`.

The next attribute is the `matchT` attribute, which has the same semantics for matching provider or consumer labels as it did in the contract for subject labels (it can take on the values of `All`, `AtLeastOne`, or `None`). This criteria applies to the provider labels as they are compared to the corresponding consumer labels. A match of the labels implies that the consumer and provider can communicate if the contract between them allows it. In other words, the contract has to allow communication and the consumer and provider labels have to match using the match criteria specified at the provider.

The consumer has no corresponding match criteria. The match type used is always determined by the provider.

Inside the provider element, `fvRsProv`, an administrator needs to specify the labels that are to be used. There are two kinds of labels, provider labels and provider subject labels. The provider labels, `vzProvLbl`, are used to match consumer labels in other EPGs that use the `matchT` criteria described earlier. The provider subject labels, `vzProvSubjLbl`, are used to match the subject labels that are specified in the contract. The only attribute of the label is its name attribute.

In the “web1” EPG, two provider subject labels, `openProv` and `secureProv`, are specified to match with the “http” and “https” subjects of the “webCtrct” contract. One provider label, “green” is specified with a match criteria of `All` that will match with the same label in the “App” EPG.

The next EPG in the example, “web2,” is very similar to “web1” except that there is only one `vzProvSubjLbl` and the labels themselves are different.

The third EPG is one called “app” and it is defined as follows:

```
<fvAEPg name="app">
  <fvRsBd tnFvBDName="solarBD1" />
  <fvRsDomAtt tDn="uni/vmmp-VMware/dom-mininet" />
  <fvRsCons tnVzBrCPName="webCtrct">
    <vzConsSubjLbl name="openCons"/>
    <vzConsSubjLbl name="secureCons"/>
    <vzConsLbl name="green"/>
  </fvRsCons>
</fvAEPg>
```

The first part is nearly the same as the “web1” EPG. The major difference is that this EPG is a consumer of the “webCtrct” and has the corresponding consumer labels and consumer subject labels. The syntax is nearly the same except that “Prov” is replaced by “Cons” in the tags. There is no match attribute in the `FvRsCons` element because the match type for matching the provider with consumer labels is specified in the provider.

In the last EPG, “db” is very similar to the “app” EPG in that it is purely a consumer.

While in this example, the EPGs were either consumers or producers of a single contract, it is typical for an EPG to be at once a producer of multiple contracts and a consumer of multiple contracts.

Closing

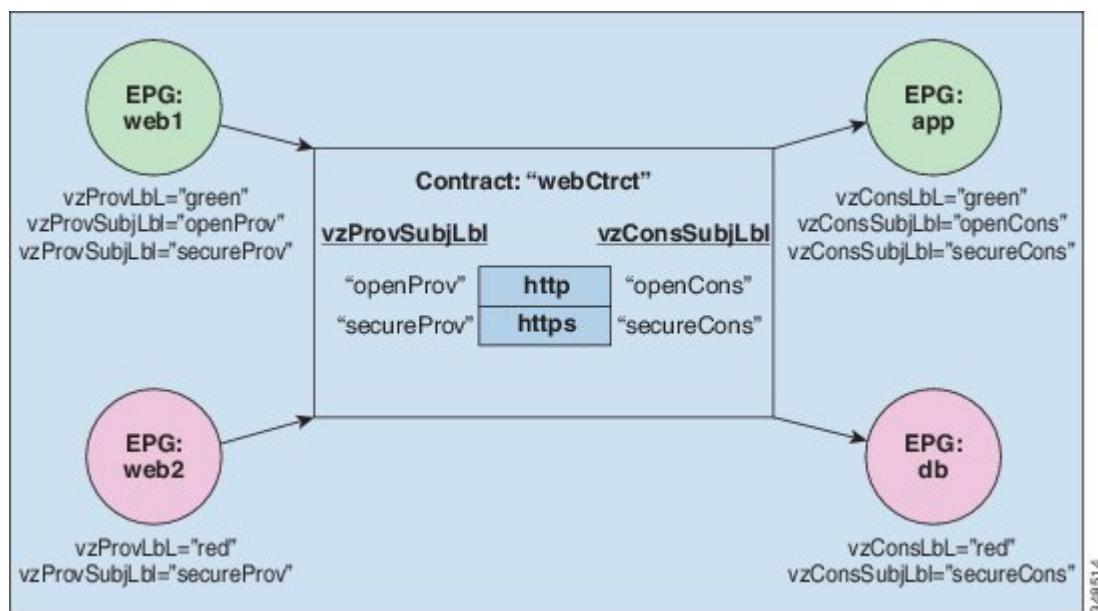
```
</fvAp>
</fvTenant>
</polUni>
```

The final few lines complete the policy.

What the Example Tenant Policy Does

The following figure shows how contracts govern endpoint group (EPG) communications.

Figure 9: Labels and Contract Determine EPG to EPG Communications



The four EPGs are named EPG:web1, EPG:web2, EPG:app, and EPG:db. EPG:web1 and EPG:web2 provide a contract called webCtrct. EPG:app and EPG:db consume that same contract.

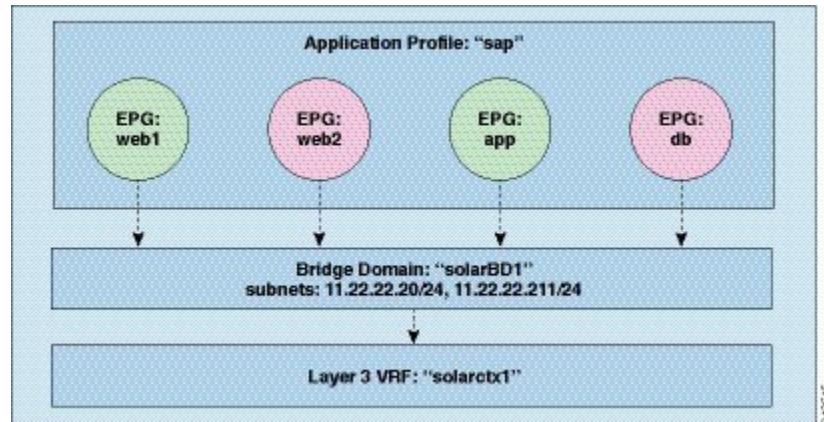
EPG:web1 can only communicate with EPG:app and EPG:web2 can only communicate with EPG:db. This interaction is controlled through the provider and consumer labels "green" and "red".

When EPG:web1 communicates with EPG:app, they use the webCtrct contract. EPG:app can initiate connections to EPG:web1 because it consumes the contract that EPG:web1 provides.

The subjects that EPG:web1 and EPG:app can use to communicate are both http and https because EPG:web1 has the provider subject label "openProv" and the http subject also has it. EPG:web1 has the provider subject label "secureProv" as does the subject https. In a similar fashion, EPG:app has subject labels "openCons" and "secureCons" that subjects http and https have.

When EPG:web2 communicates with EPG:db, they can only use the https subject because only the https subject carries the provider and consumer subject labels. EPG:db can initiate the TCP connection to EPG:web2 because EPG:db consumes the contract provided by EPG:web2.

Figure 10: Bridge Domain, Subnets, and Layer 3 VRF



The example policy specifies the relationship between EPGs, application profiles, bridge domains, and Layer 3 Virtual Routing and Forwarding (VRF) instances in the following manner: the EPGs EPG:web1, EPG:web2, EPG:app, and EPG:db are all members of the application profile called "sap."

These EPGs are also linked to the bridge domain "solarBD1." solarBD1 has two subnets, 11.22.22.XX/24 and 11.22.23.XX/24. The endpoints in the four EPGs must be within these two subnet ranges. The IP address of the default gateway in those two subnets will be 11.22.22.20 and 11.22.23.211. The solarBD1 bridge domain is linked to the "solarctx1" Layer 3 VRF.

All these policy details are contained within a tenant called "solar."

EPGs

Deploying an Application EPG through an AEP or Interface Policy Group to Multiple Ports

Through the APIC Advanced GUI and REST API, you can associate attached entity profiles directly with application EPGs. By doing so, you deploy the associated application EPGs to all those ports associated with the attached entity profile in a single configuration.

Through the APIC REST API or the NX-OS style CLI, you can deploy an application EPG to multiple ports through an Interface Policy Group.

Deploying an EPG on a Specific Port with APIC Using the REST API

Before You Begin

The tenant where you deploy the EPG is created.

Procedure

Deploy an EPG on a specific port.

Example:

```
<fvTenant name=<tenant_name> dn="uni/tn-test1" >
    <fvCtx name=<network_name> pcEnfPref="enforced" knwMcastAct="permit"/>
        <fvBD name=<bridge_domain_name> unkMcastAct="flood" >
            <fvRsCtx tnFvCtxName=<network_name>/>
        </fvBD>
    <fvAp name=<application_profile>" >
        <fvAEPg name=<epg_name>" >
            <fvRsPathAtt tDn="topology/pod-1/paths-1017/pathEp-[eth1/13]" mode="regular"
                instrImedcy="immediate" encap="vlan-20"/>
        </fvAEPg>
    </fvAp>
</fvTenant>
```

Deploying an EPG through an AEP to Multiple Interfaces Using the REST API

The interface selectors in the AEP enable you to configure multiple paths for an AEPg. The following can be selected:

- 1 A node or a group of nodes
- 2 An interface or a group of interfaces

The interfaces consume an interface policy group (and so an `infra:AttEntityP`).

- 3 The `infra:AttEntityP` is associated to the AEPg, thus specifying the VLANs to use.

An `infra:AttEntityP` can be associated with multiple AEPgs with different VLANs.

When you associate the `infra:AttEntityP` with the AEPg, as in 3, this deploys the AEPg on the nodes selected in 1, on the interfaces in 2, with the VLAN provided by 3.

In this example, the AEPg `uni/tn-Coke/ap-AP/epg-EPG1` is deployed on interfaces 1/10, 1/11, and 1/12 of nodes 101 and 102, with `vlan-102`.

Before You Begin

- Create the target application EPG (`AEPg`).
- Create the VLAN pool containing the range of VLANs you wish to use for EPG deployment with the Attached Entity Profile (AEP).
- Create the physical domain and link it to the VLAN pool and AEP.

Procedure

To deploy an AEPg on selected nodes and interfaces, send a post with XML such as the following:

Example:

```
<infraInfra dn="uni/infra">
    <infraNodeP name="NodeProfile">
        <infraLeafS name="NodeSelector" type="range">
            <infraNodeBlk name="NodeBlok" from_="101" to_="102"/>
```

```

<infraRsAccPortP tDn="uni/infra/accportprof-InterfaceProfile"/>
</infraLeafS>
</infraNodeP>

<infraAccPortP name="InterfaceProfile">
    <infraHPortS name="InterfaceSelector" type="range">
        <infraPortBlk name=" InterfaceBlock" fromCard="1" toCard="1" fromPort="10"
toPort="12"/>
        <infraRsAccBaseGrp tDn="uni/infra/funcprof/accportgrp-PortGrp" />
    </infraHPortS>
</infraAccPortP>

<infraFuncP>
    <infraAccPortGrp name="PortGrp">
        <infraRsAttEntP tDn="uni/infra/attentp-AttEntityProfile"/>
    </infraAccPortGrp>
</infraFuncP>

<infraAttEntityP name="AttEntityProfile" >
    <infraGeneric name="default" >
        <infraRsFuncToEpg tDn="uni/tn-Coke/ap-AP/epg-EPG1" encap="vlan-102"/>
    </infraGeneric>
</infraAttEntityP>
</infraInfra>

```

Creating AEP, Domains, and VLANs to Deploy an EPG on a Specific Port Using the REST API

Before You Begin

- The tenant where you deploy the EPG is already created.
- An EPG is statically deployed on a specific port.

Procedure

- Step 1** Create the interface profile, switch profile and the Attach Entity Profile (AEP).

Example:

```

<infraInfra>

    <infraNodeP name="

```

```

<infraAttEntityP name="" dn="uni/infra/attentp-<attach_entity_profile_name>">
    <infraRsDomP tDn="uni/phys-<physical_domain_name>"/>
</infraAttEntityP>

```

```
<infraInfra>
```

Step 2 Create a domain.

Example:

```

<physDomP name="" dn="uni/phys-<physical_domain_name>">
    <infraRsVlanNs tDn="uni/infra/vlanns-[<vlan_pool_name>]-static"/>
</physDomP>

```

Step 3 Create a VLAN range.

Example:

```

<fvnsVlanInstP name="" dn="uni/infra/vlanns-[<vlan_pool_name>]-static"
allocMode="static">
    <fvnsEncapBlk name="" descr="" to="vlan-25" from="vlan-10"/>
</fvnsVlanInstP>

```

Step 4 Associate the EPG with the domain.

Example:

```

<fvTenant name="" dn="uni/tn->
    <fvAEPg prio="unspecified" name="" matchT="AtleastOne"
dn="uni/tn-test1/ap-AP1/epg-<epg_name>" descr="">
        <fvRsDomAtt tDn="uni/phys-<physical_domain_name>" instrImedcy="immediate"
resImedcy="immediate"/>
    </fvAEPg>
</fvTenant>

```

Intra-EPG Isolation

Intra-EPG Isolation for Bare Metal Servers

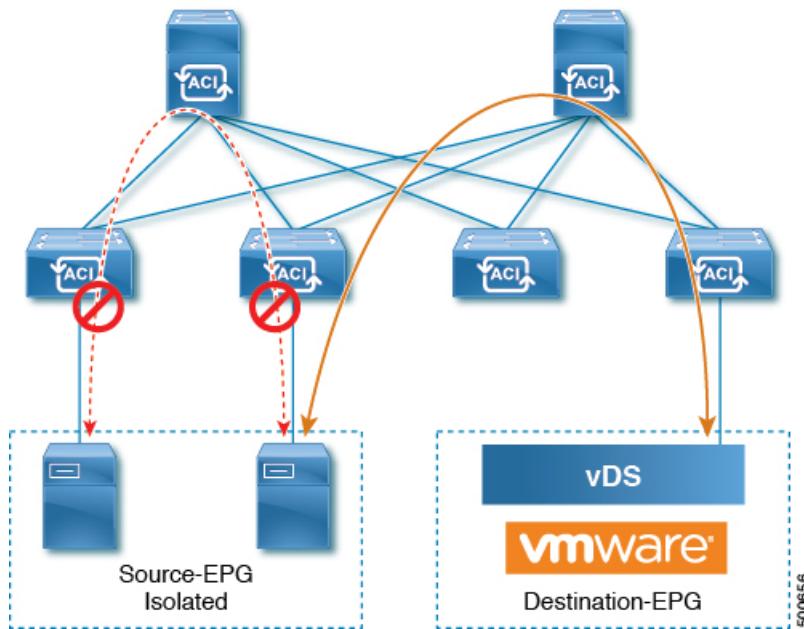
Intra-EPG endpoint isolation policies can be applied to directly connected endpoints such as bare metal servers.

Examples use cases include the following:

- Backup clients have the same communication requirements for accessing the backup service, but they don't need to communicate with each other.

- Servers behind a load balancer have the same communication requirements, but isolating them from each other protects against a server that is compromised or infected.

Figure 11: Intra-EPG Isolation for Bare Metal Servers



Bare metal EPG isolation is enforced at the leaf switch. Bare metal servers use VLAN encapsulation. All unicast, multicast and broadcast traffic is dropped (denied) within isolation enforced EPGs. ACI bridge-domains can have a mix of isolated and regular EPGs. Each Isolated EPG can have multiple VLANs where intra-vlan traffic is denied.

Configuring Intra-EPG Isolation for Bare Metal Servers Using the REST API

Before You Begin

The port the EPG uses must be associated with a bare metal server interface in the physical domain.

Procedure

Step 1 Send this HTTP POST message to deploy the application using the XML API.

Example:

```
POST https://apic-ip-address/api/mo/uni/tn-ExampleCorp.xml
```

Step 2 Include this XML structure in the body of the POST message.

Example:

```
<fvTenant name="Tenant_BareMetal" >
  <fvAp name="Web">
    <fvAEPg name="IntraEPGDeny" pcEnfPref="enforced">
      <!-- pcEnfPref="enforced" ENABLES ISOLATION-->
      <fvRsBd tnFvBDName="bd" />
```

```

<fvRsDomAtt tDn="uni/phys-Dom1" />
<!-- PATH ASSOCIATION -->
<fvRsPathAtt tDn="topology/pod-1/paths-1017/pathep-[eth1/2]" encap="vlan-51"
primaryEncap="vlan-100" instrImedcy='immediate'/>
</fvAEPg>
</fvAp>
</fvTenant>

```

Intra-EPG Isolation for VMware vDS

Intra-EPG Isolation is an option to prevent physical or virtual endpoint devices that are contained in the same base EPG or uSeg EPG from communicating with each other. By default endpoint devices included in the same EPG are allowed to communicate with one another; however, conditions exist in which total isolation of the endpoint devices from one another within an EPG is desirable. For example, if the endpoint VMs in the same EPG belong to multiple tenants, or if you want to prevent the possible spread of a virus that might infect one VM from spreading to all VMs in an EPG, intra-EPG isolation might be desirable to enforce.

An ACI virtual machine manager (VMM) domain creates an isolated PVLAN port group at the VMware vDS switch for each EPG that has intra-EPG isolation enabled. A fabric administrator specifies primary encapsulation or the fabric dynamically specifies primary encapsulation at the time of EPG-to-VMM domain association. When the fabric administrator selects the VLAN-pri and VLAN-sec values statically, the VMM domain validates that the VLAN-pri and VLAN-sec are part of a static block in the domain pool.



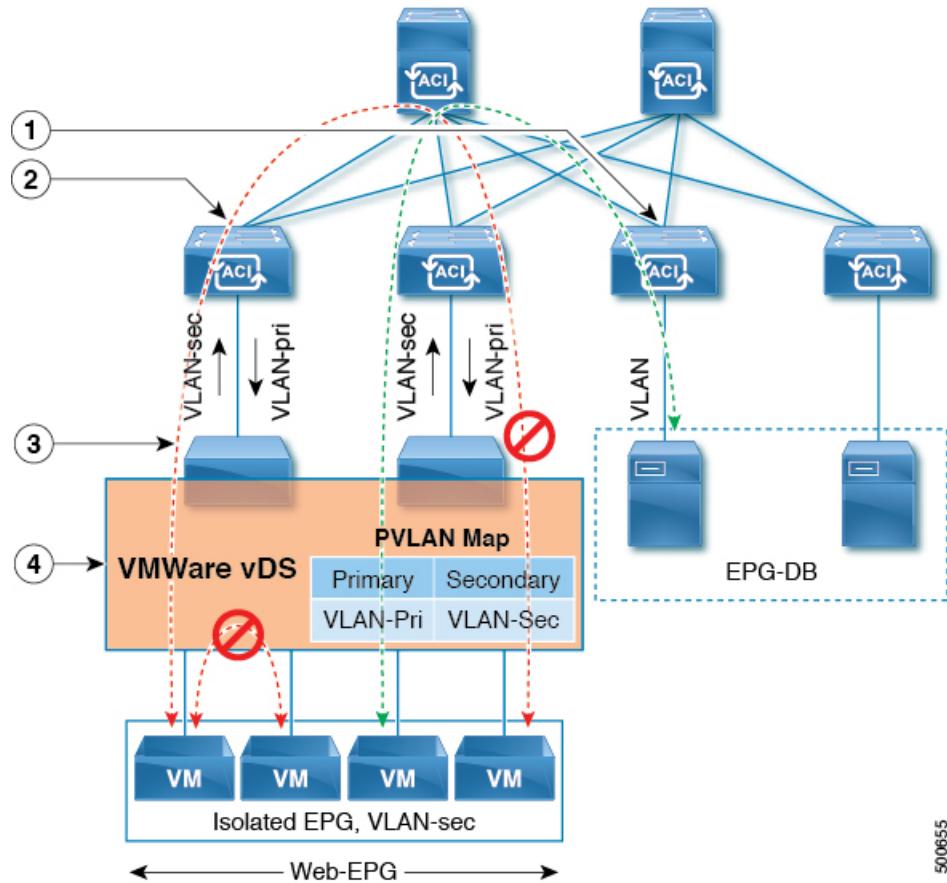
Note When intra-EPG isolation is not enforced, the VLAN-pri value is ignored even if it is specified in the configuration.

VLAN-pri/VLAN-sec pairs for the vDS switch are selected per VMM domain during the EPG-to-domain association. The port group created for the intra-EPG isolation EPGs uses the VLAN-sec tagged with type set to `PVLAN`. The vDS and fabric swap the VLAN-pri/VLAN-sec encapsulation:

- Communication from the ACI fabric to the vDS switch uses VLAN-pri.

- Communication from the vDS switch to the ACI fabric uses VLAN-sec.

Figure 12: Intra-EPG Isolation for VMware vDS



Note these details regarding this illustration:

- 1 EPG-DB sends VLAN traffic to the ACI leaf switch. The ACI egress leaf switch encapsulates traffic with a primary VLAN (PVLAN) tag and forwards it to the Web-EPG endpoint.
- 2 The vDS switch sends traffic to the ACI leaf switch using VLAN-sec. The ACI leaf switch drops all intra-EPG traffic because isolation is enforced for all intra VLAN-sec traffic within the Web-EPG.
- 3 The vDS VLAN-sec uplink to the ACI Leaf is in isolated trunk mode. The ACI leaf switch uses VLAN-pri for downlink traffic to the vDS switch.
- 4 The PVLAN map is configured in the vDS and ACI leaf switches. VM traffic from WEB-EPG is encapsulated in VLAN-sec. The vDS switch denies local intra-WEB EPG VM traffic according to the PVLAN tag. All intra-ESXi host VM traffic is sent to the ACI leaf using VLAN-sec

Related Topics

For information on configuring intra-EPG isolation in a Cisco AVS environment, see *Intra-EPG Isolation Enforcement for Cisco AVS*.

Configuring Intra-EPG Isolation for VMware vDS using the REST API

Before You Begin

Procedure

- Step 1** Send this HTTP POST message to deploy the application using the XML API.

Example:

```
POST https://apic-ip-address/api/mo/uni/tn-ExampleCorp.xml
```

- Step 2** For a VMware vDS VMM deployment, include this XML structure in the body of the POST message.

Example:

```
<fvTenant name="Tenant_VMM" >
  <fvAp name="Web">
    <fvAEPg name="IntraEPGDeny" pcEnfPref="enforced">
      <!-- pcEnfPref="enforced" ENABLES ISOLATION-->
      <fvRsBd tnFvBDName="bd" />
      <fvRsPathAtt tDn="topology/pod-1/paths-1017/pathep-[eth1/2]" encap="vlan-51"
primaryEncap="vlan-100" instrImedcy='immediate' />
      <!-- STATIC ENCAP ASSOCIATION TO VMM DOMAIN-->
      <fvRsDomAtt encap="vlan-2001" instrImedcy="lazy" primaryEncap="vlan-2002"
resImedcy="immediate" tDn="uni/vmmp-VMware/dom-DVS1">
    </fvAEPg>
  </fvAp>
</fvTenant>
```

Intra-EPG Isolation Enforcement for Cisco AVS

By default, endpoints with an EPG can communicate with each other without any contracts in place. However, beginning with Cisco AVS Release 5.2(1)SV3(1.20), you can isolate endpoints within an EPG from each other. In some instances, you might want to enforce endpoint isolation within an EPG to prevent a VM with a virus or other problem from affecting other VMs in the EPG.

You can configure isolation on endpoints within an application EPG or endpoints within Microsegmentation EPGs.

You can configure isolation on all or none of the endpoints within an application or Microsegmentation EPG; you cannot configure isolation on some endpoints but not on others.

Isolating endpoints within an EPG does not affect any contracts that enable the endpoints to communicate with endpoints in another EPG.

Isolating endpoints within an EPG will trigger a fault When the EPG is associated with Cisco AVS domains in VLAN mode.



Note

Using intra-EPG isolation on a Cisco AVS microsegment (uSeg) EPG is not currently supported. Communication will be possible between two endpoints that reside in separate uSeg EPGs if either has intra-EPG isolation enforced, regardless of any contract that exists between the two EPGs.

Configuring Intra-EPG Isolation for Cisco AVS Using the REST API

Before You Begin

Make sure that Cisco AVS is in VXLAN mode.

Procedure

-
- Step 1** Send this HTTP POST message to deploy the application using the XML API.

Example:

```
POST  
https://192.0.20.123/api/mo/uni/tn-ExampleCorp.xml
```

- Step 2** For a VMM deployment, include the XML structure in the body of the POST message.

Example:

```
Example:  
<fvTenant name="Tenant_VMM" >  
  <fvAp name="Web">  
    <fvAEPg name="IntraEPGDeny" pcEnfPref="enforced">  
      <!-- pcEnfPref="enforced" ENABLES ISOLATION-->  
      <fvRsBd tnFvBDName="bd" />  
      <fvRsDomAtt encap="vlan-2001" tDn="uni/vmmp-VMware/dom-DVS1">  
    </fvAEPg>  
  </fvAp>  
</fvTenant>
```

What to Do Next

You can select statistics and view them to help diagnose problems involving the endpoint. See the sections "Choosing Statistics to View for Isolated Endpoints" and "Viewing Statistics for Isolated Endpoints" in the *Cisco AVS Configuration Guide* or the *Cisco APIC Layer 2 Configuration Guide*.

Microsegmentation

Using Microsegmentation with Network-based Attributes on Bare Metal

You can use Cisco APIC to configure Microsegmentation with Cisco ACI to create a new, attribute-based EPG using a network-based attribute, a MAC address or one or more IP addresses. You can configure Microsegmentation with Cisco ACI using network-based attributes to isolate VMs or physical endpoints within a single base EPG or VMs or physical endpoints in different EPGs.

Using an IP-based Attribute

You can use an IP-based filter to isolate a single IP address, a subnet, or multiple of noncontiguous IP addresses in a single microsegment. You might want to isolate physical endpoints based on IP addresses as a quick and simply way to create a security zone, similar to using a firewall.

Using a MAC-based Attribute

You can use a MAC-based filter to isolate a single MAC address or multiple MAC addresses. You might want to do this if you have a server sending bad traffic into the network. By creating a microsegment with a MAC-based filter, you can isolate the server.

Configuring Microsegmentation with Cisco ACI Using the REST API

This section describes how to configure Microsegmentation with Cisco ACI for Cisco AVS or Microsoft vSwitch using the REST API.

This section describes how to configure Microsegmentation with Cisco ACI for Cisco AVS, VMware VDS or Microsoft vSwitch using the REST API.

Procedure

Step 1 Log in to the Cisco APIC.

Step 2 Post the policy to <https://apic-ip-address/api/node/mo/.xml>.

Example:

The following example configures a microsegment named 41-subnet using an IP-based attribute.

```
<polUni>
    <fvTenant dn="uni/tn-User-T1" name="User-T1">
        <fvAp dn="uni/tn-User-T1/ap-Base-EPG" name="Base-EPG">
            <fvAEPg dn="uni/tn-User-T1/ap-Base-EPG/epg-41-subnet" name="41-subnet"
isAttrBasedEPg="yes" >
                <fvRsBd tnFvBDName="BD1" />
                <fvCrtrn name="Security1">
                    <fvIpAttr name="41-filter" ip="12.41.0.0/16"/>
                </fvCrtrn>
                <fvRsDomAtt tDn="uni/vmmp-Microsoft/dom-cli-vmm1"/> / <fvRsDomAtt
tDn="uni/vmmp-VMware/dom-cli-vmm1"/>
            </fvAEPg>
        </fvAp>
    </fvTenant>
</polUni>
<polUni>
    <fvTenant dn="uni/tn-User-T1" name="User-T1">
        <fvAp dn="uni/tn-User-T1/ap-Base-EPG" name="Base-EPG">
            <fvAEPg dn="uni/tn-User-T1/ap-Base-EPG/epg-41-subnet" name="41-subnet"
pcEnfPref="enforced" isAttrBasedEPg="yes" >
                <fvRsBd tnFvBDName="BD1" />
                <fvCrtrn name="Security1">
                    <fvIpAttr name="41-filter" ip="12.41.0.0/16"/>
                </fvCrtrn>
                <fvRsDomAtt tDn="uni/vmmp-Microsoft/dom-cli-vmm1"/> / <fvRsDomAtt
tDn="uni/vmmp-VMware/dom-cli-vmm1"/>
            </fvAEPg>
        </fvAp>
    </fvTenant>
</polUni>
```

Note isAttrBasedEPg="yes" is required in Cisco APIC Release 1.2(1).

Example:

This example is for base EPG.

```
<polUni>
    <fvTenant dn="uni/tn-User-T1" name="User-T1">
        <fvAp dn="uni/tn-User-T1/ap-Base-EPG" name="Base-EPG">
```

```

> <fvAEPg dn="uni/tn-User-T1/ap-Base-EPG/baseEPG" name="baseEPG" pcEnfPref="enforced"
>   <fvRsBd tnFvBDName="BD1" />
>   <fvRsDomAtt tDn="uni/vmmp-Microsoft/dom-cli-vmm1" classPref="useg"/>
>     </fvAEPg>
>   </fvAp>
> </fvTenant>
</polUni>

```

Configuring an IP-based Microsegmented EPG as a Shared Resource Using the REST API

You can configure a microsegmented EPG with an IP-Address with 32 bit mask as a shared service, accessible by clients outside of the VRF and the current fabric.

Procedure

To configure an IP address-attribute microsegmented EPG `epg3` with a shared subnet, with an IP address and 32-bit mask, send a post with XML such as the following example. In the IP attributes, the attribute `usefvSubnet` is set to "yes."

Example:

```

<fvAEPg descr="" dn="uni/tn-t0/ap-a0/epg-epg3" fwdCtrlr="" 
         isAttrBasedEPg="yes" matchT="AtleastOne" name="epg3" pcEnfPref="unenforced" 
         prefGrMemb="exclude" prio="unspecified">
  <fvRsCons prio="unspecified" tnVzBrCPName="ip-epg"/>
  <fvRsNodeAtt descr="" encaps="unknown" instrIMedcy="immediate" mode="regular" 
    tDn="topology/pod-2/node-106"/>
  <fvSubnet ctrl="" descr="" ip="56.4.0.2/32" name="" preferred="no" 
    scope="public,shared" virtual="no"/>
  <fvRsDomAtt classPref="encap" delimiter="" encaps="unknown" encapMode="auto" 
    instrIMedcy="immediate" 
    primaryEncap="unknown" resIMedcy="immediate" tDn="uni/phys-vpc"/>
  <fvRsCustQosPol tnQosCustomPolName="" />
  <fvRsBd tnFvBDName="b2"/>
  <fvCrtrn descr="" match="any" name="default" ownerKey="" ownerTag="" prec="0">
    <fvIpAttr descr="" ip="1.1.1.3" name="ipv4" ownerKey="" ownerTag="" 
      usefvSubnet="yes"/>
  </fvCrtrn>
  <fvRsProv matchT="AtleastOne" prio="unspecified" tnVzBrCPName="ip-epg"/>
  <fvRsProv matchT="AtleastOne" prio="unspecified" tnVzBrCPName="shared-svc"/>
</fvAEPg>

```

Configuring a Network-Based Microsegmented EPG in a Bare-Metal Environment Using the REST API

This section describes how to configure network attribute microsegmentation with Cisco ACI in a bare-metal environment using the REST API.

Procedure

-
- Step 1** Log in to the Cisco APIC.
 - Step 2** Post the policy to <https://apic-ip-address/api/node/mo/.xml>.

Example:

A: The following example configures a microsegment named 41-subnet using an IP-based attribute.

```
<polUni>
    <fvTenant dn="uni/tn-User-T1" name="User-T1">
        <fvAp dn="uni/tn-User-T1/ap-Base-EPG" name="Base-EPG">
            <fvAEPg dn="uni/tn-User-T1/ap-Base-EPG/epg-41-subnet" name="41-subnet"
pcEnfPref="enforced" isAttrBasedEPg="yes" >
                <fvRsBd tnFvBDName="BD1" />
                <fvCrtrn name="Security1">
                    <fvIpAttr name="41-filter" ip="12.41.0.0/16"/>
                </fvCrtrn>
            </fvAEPg>
        </fvAp>
    </fvTenant>
</polUni>
```

Example:

This example is for base EPG for Example A: .

```
<polUni>
    <fvTenant dn="uni/tn-User-T1" name="User-T1">
        <fvAp dn="uni/tn-User-T1/ap-Base-EPG" name="Base-EPG">
            <fvAEPg dn="uni/tn-User-T1/ap-Base-EPG/baseEPG" name="baseEPG" pcEnfPref="enforced"
>
                <fvRsBd tnFvBDName="BD1" />
            </fvAEPg>
        </fvAp>
    </fvTenant>
</polUni>
```

Example:

B: The following example configures a microsegment named useg-epg using a MAC-based attribute.

```
<polUni>
    <fvTenant name="User-T1">
        <fvAp name="customer">
            <fvAEPg name="useg-epg" isAttrBasedEPg="true">
                <fvRsBd tnFvBDName="BD1"/>
                <fvRsDomAtt instrImedcy="immediate" resImedcy="immediate" tDn="uni/phys-phys"
/>
                <fvRsNodeAtt tDn="topology/pod-1/node-101" instrImedcy="immediate" />
                <fvCrtrn name="default">
                    <fvMacAttr name="mac" mac="00:11:22:33:44:55" />
                </fvCrtrn>
            </fvAEPg>
        </fvAp>
    </fvTenant>
</polUni>
```

Application Profiles

Three-Tier Application Deployment

A filter specifies the data protocols to be allowed or denied by a contract that contains the filter. A contract can contain multiple subjects. A subject can be used to realize uni- or bidirectional filters. A unidirectional filter is a filter that is used in one direction, either from consumer-to-provider (IN) or from provider-to-consumer (OUT) filter. A bidirectional filter is the same filter that is used in both directions. It is not reflexive.

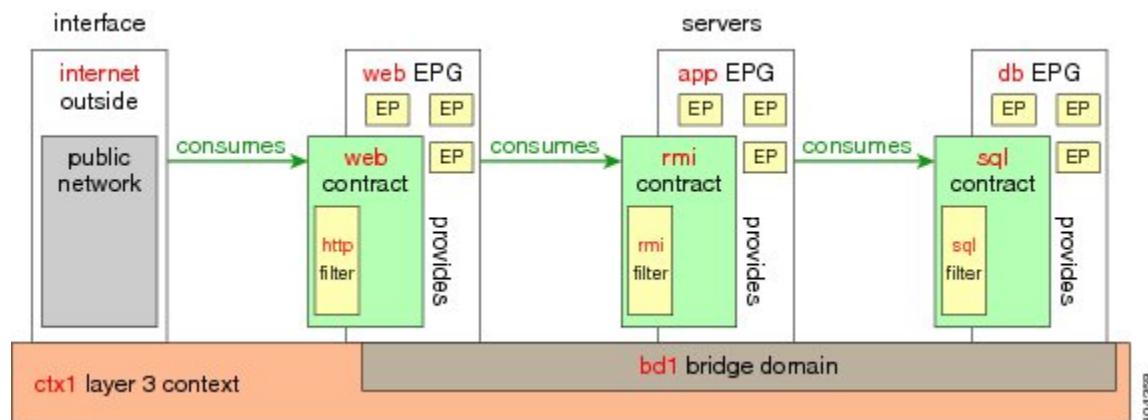
Contracts are policies that enable inter-End Point Group (inter-EPG) communication. These policies are the rules that specify communication between application tiers. If no contract is attached to the EPG, inter-EPG communication is disabled by default. No contract is required for intra-EPG communication because intra-EPG communication is always allowed.

Application profiles enable you to model application requirements that the APIC then automatically renders in the network and data center infrastructure. The application profiles enable administrators to approach the resource pool in terms of applications rather than infrastructure building blocks. The application profile is a container that holds EPGs that are logically related to one another. EPGs can communicate with other EPGs in the same application profile and with EPGs in other application profiles.

To deploy an application policy, you must create the required application profiles, filters, and contracts. Typically, the APIC fabric hosts a three-tier application within a tenant network. In this example, the application is implemented by using three servers (a web server, an application server, and a database server). See the following figure for an example of a three-tier application.

The web server has the HTTP filter, the application server has the Remote Method Invocation (RMI) filter, and the database server has the Structured Query Language (SQL) filter. The application server consumes the SQL contract to communicate with the database server. The web server consumes the RMI contract to communicate with the application server. The traffic enters from the web server and communicates with the application server. The application server then communicates with the database server, and the traffic can also communicate externally.

Figure 13: Three-Tier Application Diagram



Parameters to Create a Filter for http

The parameters to create a filter for http in this example is as follows:

Parameter Name	Filter for http
Name	http
Number of Entries	2
Entry Name	Dport-80 Dport-443

Parameter Name	Filter for http
Ethertype	IP
Protocol	tcp tcp
Destination Port	http https

Parameters to Create Filters for rmi and sql

The parameters to create filters for rmi and sql in this example are as follows:

Parameter Name	Filter for rmi	Filter for sql
Name	rmi	sql
Number of Entries	1	1
Entry Name	Dport-1099	Dport-1521
Ethertype	IP	IP
Protocol	tcp	tcp
Destination Port	1099	1521

Deploying an Application Profile Using the REST API

The port the EPG uses must belong to one of the VM Managers (VMM) or physical domains associated with the EPG.

Procedure

-
- Step 1** Send this HTTP POST message to deploy the application using the XML API.

Example:

```
POST https://apic-ip-address/api/mo/uni/tn-ExampleCorp.xml
```

- Step 2** Include this XML structure in the body of the POST message.

Example:

```
<fvTenant name="ExampleCorp">
  <fvAp name="OnlineStore">
```

```

<fvAEPg name="web">
    <fvRsBd tnFvBDName="bd1"/>
    <fvRsCons tnVzBrCPName="rmi"/>
    <fvRsProv tnVzBrCPName="web"/>
    <fvRsDomAtt tDn="uni/vmmp-VMware/dom-datacenter" delimiter=@/>
</fvAEPg>

<fvAEPg name="db">
    <fvRsBd tnFvBDName="bd1"/>
    <fvRsProv tnVzBrCPName="sql"/>
    <fvRsDomAtt tDn="uni/vmmp-VMware/dom-datacenter"/>
</fvAEPg>

<fvAEPg name="app">
    <fvRsBd tnFvBDName="bd1"/>
    <fvRsProv tnVzBrCPName="rmi"/>
    <fvRsCons tnVzBrCPName="sql"/>
    <fvRsDomAtt tDn="uni/vmmp-VMware/dom-datacenter"/>
</fvAEPg>
</fvAp>

<vzFilter name="http" >
<vzEntry dFromPort="80" name="DPort-80" prot="tcp" etherT="ip"/>
<vzEntry dFromPort="443" name="DPort-443" prot="tcp" etherT="ip"/>
</vzFilter>
<vzFilter name="rmi" >
<vzEntry dFromPort="1099" name="DPort-1099" prot="tcp" etherT="ip"/>
</vzFilter>
<vzFilter name="sql">
<vzEntry dFromPort="1521" name="DPort-1521" prot="tcp" etherT="ip"/>
</vzFilter>
    <vzBrCP name="web">
        <vzSubj name="web">
            <vzRsSubjFiltAtt tnVzFilterName="http"/>
        </vzSubj>
    </vzBrCP>

    <vzBrCP name="rmi">
        <vzSubj name="rmi">
            <vzRsSubjFiltAtt tnVzFilterName="rmi"/>
        </vzSubj>
    </vzBrCP>

    <vzBrCP name="sql">
        <vzSubj name="sql">
            <vzRsSubjFiltAtt tnVzFilterName="sql"/>
        </vzSubj>
    </vzBrCP>
</fvTenant>

```

In the string fvRsDomAtt tDn="uni/vmmp-VMware/dom-datacenter" delimiter=@/, **delimiter=@** is optional. If you do not enter a delimiter, the system will use the default | delimiter.

In the XML structure, the first line modifies, or creates if necessary, the tenant named ExampleCorp.

```
<fvTenant name="ExampleCorp">
```

This line creates an application network profile named OnlineStore.

```
<fvAp name="OnlineStore">
```

The elements within the application network profile create three endpoint groups, one for each of the three servers. The following lines create an endpoint group named web and associate it with an existing bridge domain named bd1. This endpoint group is a consumer, or destination, of the traffic allowed by the binary

contract named rmi and is a provider, or source, of the traffic allowed by the binary contract named web. The endpoint group is associated with the VMM domain named datacenter.

```
<fvAEPg name="web">
  <fvRsBd tnFvBDName="bd1"/>
  <fvRsCons tnVzBrCPName="rmi"/>
  <fvRsProv tnVzBrCPName="web"/>
  <fvRsDomAtt tDn="uni/vmmp-VMware/dom-datacenter"/>
</fvAEPg>
```

The remaining two endpoint groups, for the application server and the database server, are created in a similar way.

The following lines define a traffic filter named http that specifies TCP traffic of types HTTP (port 80) and HTTPS (port 443).

```
<vzFilter name="http" >
  <vzEntry dFromPort="80" name="DPort-80" prot="tcp" etherT="ip"/>
  <vzEntry dFromPort="443" name="DPort-443" prot="tcp" etherT="ip"/>
</vzFilter>
```

The remaining two filters, for application data and database (sql) data, are created in a similar way.

The following lines create a binary contract named web that incorporates the filter named http:

```
<vzBrCP name="web">
  <vzSubj name="web">
    <vzRsSubjFiltAtt tnVzFilterName="http"/>
  </vzSubj>
</vzBrCP>
```

The remaining two contracts, for rmi and sql data protocols, are created in a similar way.

The final line closes the structure:

```
</fvTenant>
```

Contracts, Taboo Contracts, and Preferred Groups

Security Policy Enforcement

As traffic enters the leaf switch from the front panel interfaces, the packets are marked with the EPG of the source EPG. The leaf switch then performs a forwarding lookup on the packet destination IP address within the tenant space. A hit can result in any of the following scenarios:

- 1 A unicast (/32) hit provides the EPG of the destination endpoint and either the local interface or the remote leaf switch VTEP IP address where the destination endpoint is present.
- 2 A unicast hit of a subnet prefix (not /32) provides the EPG of the destination subnet prefix and either the local interface or the remote leaf switch VTEP IP address where the destination subnet prefix is present.
- 3 A multicast hit provides the local interfaces of local receivers and the outer destination IP address to use in the VXLAN encapsulation across the fabric and the EPG of the multicast group.

**Note**

Multicast and external router subnets always result in a hit on the ingress leaf switch. Security policy enforcement occurs as soon as the destination EPG is known by the ingress leaf switch.

A miss result in the forwarding table causes the packet to be sent to the forwarding proxy in the spine switch. The forwarding proxy then performs a forwarding table lookup. If it is a miss, the packet is dropped. If it is a hit, the packet is sent to the egress leaf switch that contains the destination endpoint. Because the egress leaf switch knows the EPG of the destination, it performs the security policy enforcement. The egress leaf switch must also know the EPG of the packet source. The fabric header enables this process because it carries the EPG from the ingress leaf switch to the egress leaf switch. The spine switch preserves the original EPG in the packet when it performs the forwarding proxy function.

On the egress leaf switch, the source IP address, source VTEP, and source EPG information are stored in the local forwarding table through learning. Because most flows are bidirectional, a return packet populates the forwarding table on both sides of the flow, which enables the traffic to be ingress filtered in both directions.

Contracts and Taboo Contracts

Contracts Contain Security Policy Specifications

In the ACI security model, contracts contain the policies that govern the communication between EPGs. The contract specifies what can be communicated and the EPGs specify the source and destination of the communications. Contracts link EPGs, as shown below.

EPG 1 ----- CONTRACT ----- EPG 2

Endpoints in EPG 1 can communicate with endpoints in EPG 2 and vice versa if the contract allows it. This policy construct is very flexible. There can be many contracts between EPG 1 and EPG 2, there can be more than two EPGs that use a contract, and contracts can be reused across multiple sets of EPGs, and more.

There is also directionality in the relationship between EPGs and contracts. EPGs can either provide or consume a contract. An EPG that provides a contract is typically a set of endpoints that provide a service to a set of client devices. The protocols used by that service are defined in the contract. An EPG that consumes a contract is typically a set of endpoints that are clients of that service. When the client endpoint (consumer) tries to connect to a server endpoint (provider), the contract checks to see if that connection is allowed. Unless otherwise specified, that contract would not allow a server to initiate a connection to a client. However, another contract between the EPGs could easily allow a connection in that direction.

This providing/consuming relationship is typically shown graphically with arrows between the EPGs and the contract. Note the direction of the arrows shown below.

EPG 1 <-----consumes----- CONTRACT <-----provides----- EPG 2

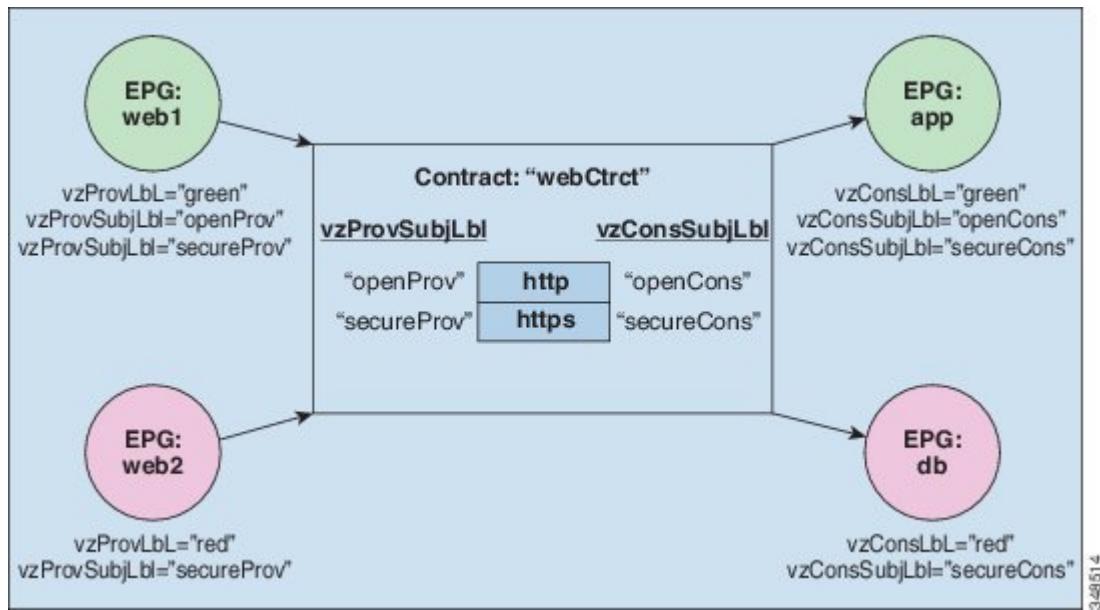
The contract is constructed in a hierarchical manner. It consists of one or more subjects, each subject contains one or more filters, and each filter can define one or more protocols.

Figure 14: Contract Filters



The following figure shows how contracts govern EPG communications.

Figure 15: Contracts Determine EPG to EPG Communications



For example, you may define a filter called HTTP that specifies TCP port 80 and port 8080 and another filter called HTTPS that specifies TCP port 443. You might then create a contract called webCtrct that has two sets of subjects. openProv and openCons are the subjects that contain the HTTP filter. secureProv and secureCons are the subjects that contain the HTTPS filter. This webCtrct contract can be used to allow both secure and non-secure web traffic between EPGs that provide the web service and EPGs that contain endpoints that want to consume that service.

These same constructs also apply for policies that govern virtual machine hypervisors. When an EPG is placed in a virtual machine manager (VMM) domain, the APIC downloads all of the policies that are associated with the EPG to the leaf switches with interfaces connecting to the VMM domain. For a full explanation of VMM domains, see the *Virtual Machine Manager Domains* chapter of *Application Centric Infrastructure Fundamentals*. When this policy is created, the APIC pushes it (pre-populates it) to a VMM domain that specifies which switches allow connectivity for the endpoints in the EPGs. The VMM domain defines the set

of switches and ports that allow endpoints in an EPG to connect to. When an endpoint comes on-line, it is associated with the appropriate EPGs. When it sends a packet, the source EPG and destination EPG are derived from the packet and the policy defined by the corresponding contract is checked to see if the packet is allowed. If yes, the packet is forwarded. If no, the packet is dropped.

Contracts consist of 1 or more subjects. Each subject contains 1 or more filters. Each filter contains 1 or more entries. Each entry is equivalent to a line in an Access Control List (ACL) that is applied on the Leaf switch to which the endpoint within the endpoint group is attached.

In detail, contracts are comprised of the following items:

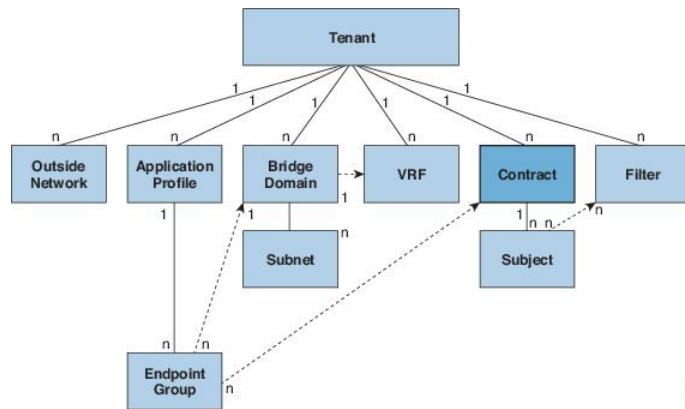
- Name—All contracts that are consumed by a tenant must have different names (including contracts created under the common tenant or the tenant itself).
- Subjects—A group of filters for a specific application or service.
- Filters—Used to classify traffic based upon layer 2 to layer 4 attributes (such as Ethernet type, protocol type, TCP flags and ports).
- Actions—Action to be taken on the filtered traffic. The following actions are supported:
 - Permit the traffic (regular contracts, only)
 - Mark the traffic (DSCP/CoS) (regular contracts, only)
 - Redirect the traffic (regular contracts, only, through a service graph)
 - Copy the traffic (regular contracts, only, through a service graph or SPAN)
 - Block the traffic (taboo contracts, only)
 - Log the traffic (taboo contracts, only)
- Aliases—(Optional) A changeable name for an object. Although the name of an object, once created, cannot be changed, the Alias is a property that can be changed.

Thus, the contract allows more complex actions than just allow or deny. The contract can specify that traffic that matches a given subject can be re-directed to a service, can be copied, or can have its QoS level modified. With pre-population of the access policy in the concrete model, endpoints can move, new ones can come on-line, and communication can occur even if the APIC is off-line or otherwise inaccessible. The APIC is removed from being a single point of failure for the network. Upon packet ingress to the ACI fabric, security policies are enforced by the concrete model running in the switch.

Contracts

In addition to EPGs, contracts (`vzBrCP`) are key objects in the policy model. EPGs can only communicate with other EPGs according to contract rules. The following figure shows the location of contracts in the management information tree (MIT) and their relation to other objects in the tenant.

Figure 16: Contracts



An administrator uses a contract to select the type(s) of traffic that can pass between EPGs, including the protocols and ports allowed. If there is no contract, inter-EPG communication is disabled by default. There is no contract required for intra-EPG communication; intra-EPG communication is always implicitly allowed.

You can also configure contract preferred groups that enable greater control of communication between EPGs in a VRF. If most of the EPGs in the VRF should have open communication, but a few should only have limited communication with the other EPGs, you can configure a combination of a contract preferred group and contracts with filters to control communication precisely.

Contracts govern the following types of endpoint group communications:

- Between ACI fabric application EPGs (`fvAEPg`), both intra-tenant and inter-tenant



Note In the case of a shared service mode, a contract is required for inter-tenant communication. A contract is used to specify static routes across VRFs, even though the tenant VRF does not enforce a policy.

- Between ACI fabric application EPGs and Layer 2 external outside network instance EPGs (`l2extInstP`)
- Between ACI fabric application EPGs and Layer 3 external outside network instance EPGs (`l3extInstP`)
- Between ACI fabric out-of-band (`mgmtOOB`) or in-band (`mgmtInB`) management EPGs

Contracts govern the communication between EPGs that are labeled providers, consumers, or both. EPG providers expose contracts with which a would-be consumer EPG must comply. The relationship between an EPG and a contract can be either a provider or consumer. When an EPG provides a contract, communication with that EPG can be initiated from other EPGs as long as the communication complies with the provided contract. When an EPG consumes a contract, the endpoints in the consuming EPG may initiate communication with any endpoint in an EPG that is providing that contract.

**Note**

An EPG can both provide and consume the same contract. An EPG can also provide and consume multiple contracts simultaneously.

Configuring a Contract Using the REST API

Procedure

Configure a contract using an XML POST request similar to the following example:

Example:

```
<fvAp name="VRF64_app_prof" prio="unspecified" ownerTag="" ownerKey=""
dn="uni/tn-Tenant64/ap-VRF64_app_prof" descr="">
<fvRsApMonPol tnMonEPGPolName="default"/>
<fvAEPg name="EPG64" prio="unspecified" descr="" prefGrMemb="exclude" pcEnfPref="unenforced"
matchT="AtleastOne" isAttrBasedEPg="no" fwdCtrl="">
<fvRsPathAtt descr="" tDn="topology/pod-1/paths-102/pathep-[eth1/1]" primaryEncap="unknown"
mode="regular" instrImedcy="lazy" encap="vlan-21"/>
<fvRsDomAtt tDn="uni/phys-phys" primaryEncap="unknown" instrImedcy="lazy" encap="unknown"
resImedcy="lazy" encapMode="auto" delimiter="" classPref="encap"/>
<fvRsBd tnFvBDName="VRF64-BD"/>
<fvRsCustQosPol tnQosCustomPolName="" />
</fvAEPg>
</fvAp>
```

Configuring a Taboo Contract Using the REST API

Before You Begin

The following objects must be created:

- The tenant that will be associated with this **Taboo Contract**
- An application profile for the tenant
- At least one EPG for the tenant

Procedure

To create a taboo contract with the REST API, use XML such as in the following example:

Example:

```
<vzTaboo ownerTag="" ownerKey="" name="VRF64_Taboo_Contract"
dn="uni/tn-Tenant64/taboo-VRF64_Taboo_Contract" descr=""><vzTSubj
name="EPG_subject" descr=""><vzRsDenyRule tnVzFilterName="default"
directives="log"/>
</vzTSubj>
</vzTaboo>
```

Contract Preferred Groups

About Contract Preferred Groups

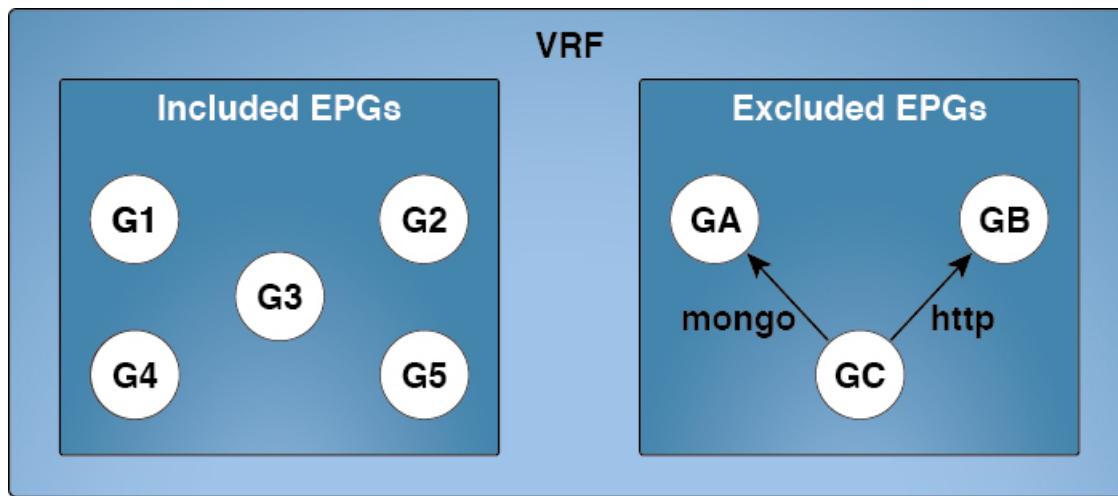
There are two types of policy enforcements available for EPGs in a VRF with a contract preferred group configured:

- Included EPGs: EPGs can freely communicate with each other without contracts, if they have membership in a contract preferred group. This is based on the source-any-destination-any-permit default rule.
- Excluded EPGs: EPGs that are not members of preferred groups require contracts to communicate with each other. Otherwise, the default source-any-destination-any-deny rule applies.

The contract preferred group feature enables greater control of communication between EPGs in a VRF. If most of the EPGs in the VRF should have open communication, but a few should only have limited communication with the other EPGs, you can configure a combination of a contract preferred group and contracts with filters to control intra-EPG communication precisely.

EPGs that are excluded from the preferred group can only communicate with other EPGs if there is a contract in place to override the source-any-destination-any-deny default rule.

Figure 17: Contract Preferred Group Overview



Source	Destination	Filter	Action
any	any	implicit	permit
GA	any	implicit	deny
any	GA	implicit	deny
GB	any	implicit	deny
any	GB	implicit	deny
GC	any	implicit	deny
any	GC	implicit	deny

Source	Destination	Filter	Action
GC	GA	mongo	permit
GA	GC	mongo	permit
GC	GB	http	permit
GB	GC	http	permit

501168

Configuring Contract Preferred Groups Using the REST API

The following example creates a contract preferred group in `vrf64`, and creates three EPGs in the VRF:

- `epg-ldap`—Included in the preferred group
- `mail`—Included in the preferred group
- `radius`—Excluded from the preferred group

Before You Begin

Create the tenants, VRFs, and the EPGs in the VRF.

Procedure

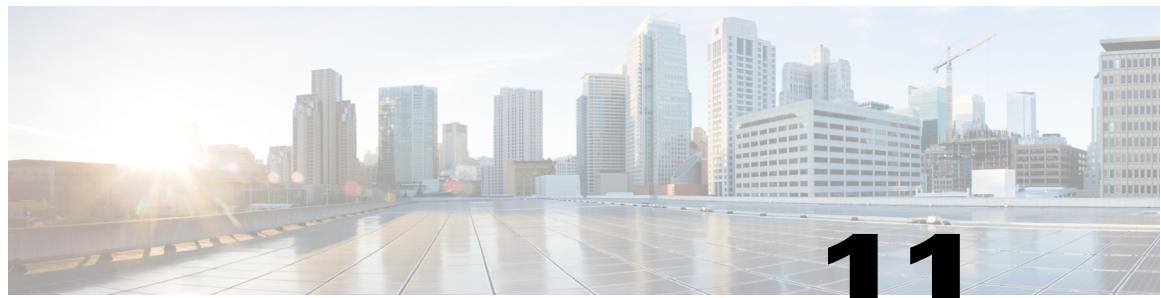
Create a contract preferred group by sending a post, with XML such as the example:

Example:

```
<polUni>
  <fvTenant name="tenant64">
    <fvCtx name="vrf64"> <vzAny prefGrMemb="enabled"/> </fvCtx>
    <fvBD name="bd64"> <fvRsCtx tnFvCtxName="vrf64"/> </fvBD>
    <fvAp name="app-ldap">
      <fvAEPg name="epg-ldap" prefGrMemb="include">
        <fvRsBd tnFvBDName="bd64"/>
        <fvRsPathAtt tDn="topology/pod-1/paths-101/pathep-[eth1/3]" encap="vlan-113"
instrImedcy="immediate"/>
      </fvAEPg>
      <fvAEPg name="mail" prefGrMemb="include">
        <fvRsBd tnFvBDName="bd64"/>
        <fvRsPathAtt tDn="topology/pod-1/paths-101/pathep-[eth1/4]" encap="vlan-114"
instrImedcy="immediate"/>
      </fvAEPg>
      <fvAEPg name="radius" prefGrMemb="exclude">
        <fvRsBd tnFvBDName="bd64"/>
        <fvRsPathAtt tDn="topology/pod-1/paths-101/pathep-[eth1/5]" encap="vlan-115"
instrImedcy="immediate"/>
      </fvAEPg>
    </fvAp>
  </fvTenant>
</polUni>
```

What to Do Next

Create a contract governing the communication of the `radius` EPG with other EPGs.



CHAPTER 11

Provisioning Core Services

- [DHCP](#), page 157
- [DNS](#), page 160
- [NTP](#), page 162
- [Tetration](#), page 163
- [NetFlow](#), page 164
- [ACL Contract Permit and Deny Logging](#), page 167
- [DOM Statistics](#), page 168
- [Syslog](#), page 170
- [Data Plane Policing](#), page 172
- [Traffic Storm Control](#), page 175

DHCP

Configuring a DHCP Relay Policy

A DHCP relay policy may be used when the DHCP client and server are in different subnets. If the client is on an ESX hypervisor with a deployed vShield Domain profile, then the use of a DHCP relay policy configuration is mandatory.

When a vShield controller deploys a Virtual Extensible Local Area Network (VXLAN), the hypervisor hosts create a kernel (vmkN, virtual tunnel end-point [VTEP]) interface. These interfaces need an IP address in the infrastructure tenant that uses DHCP. Therefore, you must configure a DHCP relay policy so that the APIC can act as the DHCP server and provide these IP addresses.

When an ACI fabric acts as a DHCP relay, it inserts the DHCP Option 82 (the DHCP Relay Agent Information Option) in DHCP requests that it proxies on behalf of clients. If a response (DHCP offer) comes back from a DHCP server without Option 82, it is silently dropped by the fabric. Therefore, when the ACI fabric acts as a DHCP relay, DHCP servers providing IP addresses to compute nodes attached to the ACI fabric must support Option 82.

Configuring a DHCP Server Policy for the APIC Infrastructure Using the REST API

- This task is a prerequisite for users who want to create a vShield Domain Profile.
- The port and the encapsulation used by the application EPG must belong to a physical or VM Manager (VMM) domain. If no such association with a domain is established, the APIC continues to deploy the EPG but raises a fault.
- Cisco APIC supports DHCP relay for both IPv4 and IPv6 tenant subnets. DHCP server addresses can be IPv4 or IPv6. DHCPv6 relay will occur only if IPv6 is enabled on the fabric interface and one or more DHCPv6 relay servers are configured.

Before You Begin

Make sure that Layer 2 or Layer 3 management connectivity is configured.

Procedure

Configure the APIC as the DHCP server policy for the infrastructure tenant.

Note This relay policy will be pushed to all the leaf ports that are connected hypervisors using the attach entity profile configuration. For details about configuring with attach entity profile, see the examples related to creating VMM domain profiles.

Example:

```
<!-- api/policymgr/mo/.xml -->
<polUni>

POST https://apic-ip-address/api/mo/uni.xml

<fvTenant name="infra">

<dhcpRelayP name="DhcpRelayP" owner="tenant">
    <dhcpRsProv tDn="uni/tn-infra/ap-access/epg-default" addr="10.0.0.1" />
</dhcpRelayP>

<fvBD name="default">
    <dhcpLbl name="DhcpRelayP" owner="tenant"/>
</fvBD>

</fvTenant>
</polUni>
```

Layer 2 and Layer 3 DHCP Relay Sample Policies

This sample policy provides an example of a consumer tenant L3extOut DHCP relay configuration:

```
<polUni>
    <!-- Consumer Tenant 2 -->
    <fvTenant
        dn="uni/tn-tenant1"
        name="tenant1">
        <fvCtx name="dhcp"/>

        <!-- DHCP client bridge domain -->
        <fvBD name="cons2">
            <fvRsBDToOut tnL3extOutName='L3OUT' />
            <fvRsCtx tnFvCtxName="dhcp" />
            <fvSubnet ip="20.20.20.1/24"/>
            <dhcpLbl name="DhcpRelayP" owner="tenant"/>
        </fvBD>
    </fvTenant>
</polUni>
```

```

</fvBD>
<!-- L3Out EPG DHCP -->
<l3extOut name="L3OUT">
    <l3extRsEctx tnFvCtxName="dhcp"/>
    <l3extInstP name="l3extInstP-1">
        <!-- Allowed routes to L3out to send traffic -->
        <l3extSubnet ip="100.100.100.0/24" />
    </l3extInstP>
    <l3extLNodeP name="l3extLNodeP-pc">
        <!-- VRF External loopback interface on node -->
        <l3extRsNodeL3OutAtt
            tDn="topology/pod-1/node-1018"
            rtrId="10.10.10.1" />
        <l3extLIfP name='l3extLIfP-pc'>
            <l3extRsPathL3OutAtt
                tDn="topology/pod-1/paths-1018/pathep-[eth1/7]"
                encap='vlan-900'
                ifInstT='sub-interface'
                addr="100.100.100.50/24"
                mtu="1500"/>
            </l3extLIfP>
        </l3extLNodeP>
    </l3extOut>
<!-- Static DHCP Client Configuration -->
<fvAp name="cons2">
    <fvAEPg name="APP">
        <fvRsBd tnFvBDName="cons2"/>
        <fvRsDomAtt tDn="uni/phys-mininet"/>
        <fvRsPathAtt
            tDn="topology/pod-1/paths-1017/pathep-[eth1/3]"
            encap="vlan-1000"
            instrImedcy='immediate'
            mode='native' />
    </fvAEPg>
</fvAp>
<!-- DHCP Server Configuration -->
<dhcpRelayP
    name="DhcpRelayP"
    owner="tenant"
    mode="visible">
    <dhcpRsProv
        tDn="uni/tn-tenant1/out-L3OUT/instP-l3extInstP-1"
        addr="100.100.100.1"/>
</dhcpRelayP>
</fvTenant>
</polUni>

```

This sample policy provides an example of a consumer tenant L2extOut DHCP relay configuration:

```

<fvTenant
    dn="uni/tn-dhcp12Out"
    name="dhcp12Out">
    <fvCtx name="dhcp12Out"/>
        <!-- bridge domain -->

    <fvBD name="provBD">
        <fvRsCtx tnFvCtxName="dhcp12Out" />
        <fvSubnet ip="100.100.100.50/24" scope="shared"/>
    </fvBD>

        <!-- Consumer bridge domain -->
    <fvBD name="cons2">
        <fvRsCtx tnFvCtxName="dhcp12Out" />
        <fvSubnet ip="20.20.20.1/24"/>
        <dhcpLbl name="DhcpRelayP" owner="tenant"/>
    </fvBD>

        <vzFilter name='t0f0' >
            <vzEntry name='t0f0e9'></vzEntry>
        </vzFilter>

        <vzBrCP name="webCtrct" scope="global">

```

```

<vzSubj name="app">
    <vzRsSubjFiltAtt tnVzFilterName="t0f0"/>
</vzSubj>
</vzBrCP>

<l2extOut name="l2Out">
    <l2extLNodeP name='l2ext'>
        <l2extLIfP name='l2LifP'>
            <l2extRsPathL2OutAtt tDn="topology/pod-1/paths-1018/pathEP-[eth1/7]" />
        </l2extLIfP>
        <l2extInstP name='l2inst'>
            <fvRsProv tnVzBrCPName="webCtrct"/>
        </l2extInstP>
    <l2extRsEBd tnFvBDName="provBD" encap='vlan-900' />
</l2extOut>

<fvAp name="cons2">
    <fvAEPg name="APP">
        <fvRsBd tnFvBDName="cons2" />
        <fvRsDomAtt tDn="uni/phys-mininet" />
        <fvRsBd tnFvBDName="SolarBD2" />
        <fvRsPathAtt tDn="topology/pod-1/paths-1018/pathEP-[eth1/48]" encap="vlan-1000" instrImedcy='immediate' mode='native' />
    </fvAEPg>
</fvAp>
    <dhcpRelayP name="DhcpRelayP" owner="tenant" mode="visible">
        <dhcpRsProv tDn="uni/tn-dhcp12Out/l2out-l2Out/instP-l2inst" addr="100.100.100.1" />
    </dhcpRelayP>
</fvTenant>

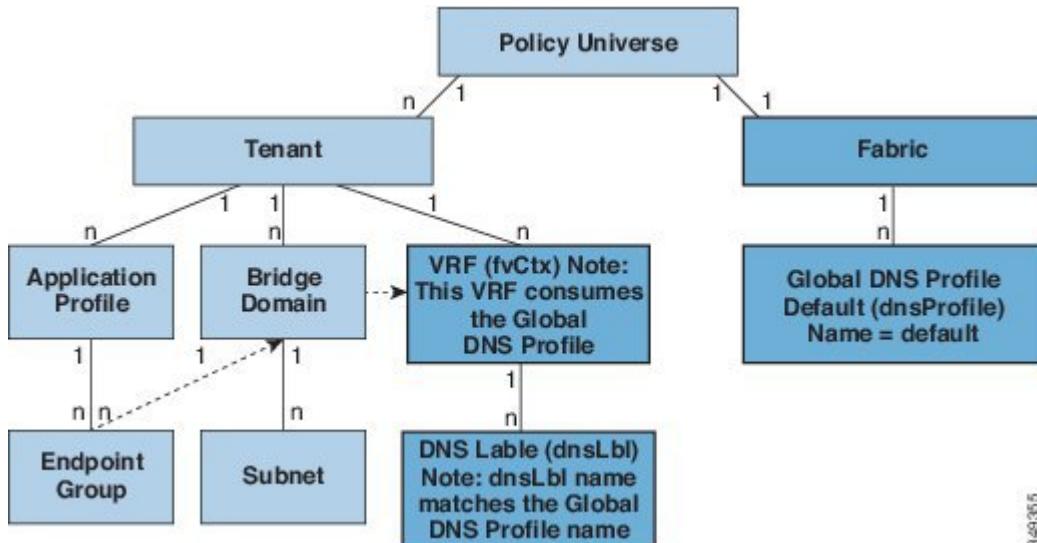
```

DNS

DNS

The ACI fabric DNS service is contained in the fabric managed object. The fabric global default DNS profile can be accessed throughout the fabric. The figure below shows the logical relationships of the DNS-managed objects within the fabric.

Figure 18: DNS



349355

A VRF (context) must contain a `dnsLBL` object in order to use the global default DNS service. Label matching enables tenant VRFs to consume the global DNS provider. Because the name of the global DNS profile is "default," the VRF label name is "default" (`dnsLBL name = default`).

Configuring a DNS Service Policy to Connect with DNS Providers Using the REST API

Before You Begin

Make sure that Layer 2 or Layer 3 management connectivity is configured.

Procedure

- Step 1** Configure the DNS service policy.

Example:

```
POST URL :  
https://apic-IP-address/api/node/mo/uni/fabric.xml  
  
<dnsProfile name="default">  
    <dnsProv addr="172.21.157.5" preferred="yes"/>  
    <dnsProv addr="172.21.157.6"/>  
    <dnsDomain name="cisco.com" isDefault="yes"/>  
    <dnsRsProfileToEpg tDn="uni/tn-mgmt/mgmtp-default/oob-default"/>  
</dnsProfile>
```

- Step 2** Configure the DNS label under the out-of-band management tenant.

Example:

```
POST URL: https://apic-IP-address/api/node/mo/uni/tn-mgmt/ctx-oob.xml  
<dnsLbl name="default" tag="yellow-green"/>
```

DNS Policy Example

This sample policy creates a DNS profile and associates it with a tenant.

Create the DNS profile:

```
<!-- /api/policymgr/mo/.xml -->  
<polUni>  
<fabricInst>  
    <dnsProfile name="default">  
        <dnsProv addr="172.21.157.5" preferred="yes"/>  
        <dnsDomain name="insieme.local" isDefault="yes"/>  
        <dnsRsProfileToEpg tDn="uni/tn-mgmt/mgmtp-default/oob-default"/>  
    </dnsProfile>  
</fabricInst>  
</polUni>
```

Associate the profile with the tenant that will consume it:

```
<!-- /api/policymgr/mo/.xml -->  
<polUni>  
    <fvTenant name='t1'>  
        <fvCtx name='ctx0'>  
            <dnsLbl name='default'/>  
        </fvCtx>
```

```
</fvTenant>
</polUni>
```

NTP

Time Synchronization and NTP

Within the Cisco Application Centric Infrastructure (ACI) fabric, time synchronization is a crucial capability upon which many of the monitoring, operational, and troubleshooting tasks depend. Clock synchronization is important for proper analysis of traffic flows as well as for correlating debug and fault time stamps across multiple fabric nodes.

An offset present on one or more devices can hamper the ability to properly diagnose and resolve many common operational issues. In addition, clock synchronization allows for the full utilization of the atomic counter capability that is built into the ACI upon which the application health scores depend. Nonexistent or improper configuration of time synchronization does not necessarily trigger a fault or a low health score. You should configure time synchronization before deploying a full fabric or applications so as to enable proper usage of these features. The most widely adapted method for synchronizing a device clock is to use Network Time Protocol (NTP).

Prior to configuring NTP, consider what management IP address scheme is in place within the ACI fabric. There are two options for configuring management of all ACI nodes and Application Policy Infrastructure Controllers (APICs), in-band management and/or out-of-band management. Depending upon which management option is chosen for the fabric, configuration of NTP will vary. Another consideration in deploying time synchronization is where the time source is located. The reliability of the source must be carefully considered when determining if you will use a private internal clock or an external public clock.

Configuring NTP Using the REST API

Procedure

Step 1 Configure NTP.

Example:

```
POST url: https://APIC-IP/api/node/mo/uni/fabric/time-test.xml
```

```
<imdata totalCount="1">
    <datetimePol adminSt="enabled" authSt="disabled" descr="" dn="uni/fabric/time-CiscoNTPPol"
        name="CiscoNTPPol" ownerKey="" ownerTag="">
        <datetimeNtpProv descr="" keyId="0" maxPoll="6" minPoll="4" name="10.10.10.11"
            preferred="yes">
            <datetimeRsNtpProvToEpg tDn="uni/tn-mgmt/mgmtp-default/inb-default"/>
        </datetimeNtpProv>
    </datetimePol>
</imdata>
```

Step 2 Add the default Date Time Policy to the pod policy group.

Example:

```
POST url: https://APIC-IP/api/node/mo/uni/fabric/funcprof/podpgrp-cal01/rsTimePol.xml
```

```
POST payload: <imdata totalCount="1">
    <fabricRsTimePol tnDatetimePolName="CiscoNTPPol">
```

```
</fabricRsTimePol>
</imdata>
```

- Step 3** Add the pod policy group to the default pod profile.

Example:

```
POST url:
https://APIC-IP/api/node/mo/uni/fabric/podprof-default/pods-default-typ-ALL/rspodPGrp.xml

payload: <imdata totalCount="1">
<fabricRsPodPGrp tDn="uni/fabric/funcprof/podpgrp-cal01" status="created">
</fabricRsPodPGrp>
</imdata>
```

Tetration

Overview

This article provides examples of how to configure Cisco Tetration Analytics when using the Cisco APIC. The following information applies when configuring Cisco Tetration Analytics.

- An inband management IP address must be configured on each leaf where the Cisco Tetration Analytics agent is active.
- Define an analytics policy and specify the destination IP address of the Cisco Tetration Analytics server.
- Create a switch profile and include the policy group created in the previous step.

Configuring Cisco Tetration Analytics Using the REST API

Procedure

- Step 1** Create the analytics policy.

Example:

```
<analyticsCluster name="tetration" >
<analyticsCfgSrv name="srv1" ip="10.30.30.7" >
</analyticsCfgSrv>
</analyticsCluster>
```

- Step 2** Associate analytics with the policy group.

Example:

```
<fabricLeNodePGrp descr="" name="mypolicy6" ownerKey="" ownerTag="" rn="lenodepgrp-mypolicy6"
status="">
<fabricRsNodeCfgSrv rn="rsnodeProv" status="" tDn="uni/fabric/analytics/cluster-tetration/cfgsrv-srv1" />
</fabricLeNodePGrp>
```

- Step 3** Associate the policy group with the switch.

Example:

```
<fabricLeafP name="leafs" rn="leprof-leafs" status="" >
    <fabricLeafS name="sw" rn="leaves-sw-typ-range" status="">
        <fabricRsLeNodePGrp rn="rsleNodePGrp" tDn="uni/fabric/funcprof/lenodepgrp-mypolicy6"/>
            <fabricNodeBlk name="switches" from_="101" to_="101" />
        </fabricLeafS>
    </fabricLeafP>
```

NetFlow

About NetFlow

The NetFlow technology provides the metering base for a key set of applications, including network traffic accounting, usage-based network billing, network planning, as well as denial of services monitoring, network monitoring, outbound marketing, and data mining for both service providers and enterprise customers. Cisco provides a set of NetFlow applications to collect NetFlow export data, perform data volume reduction, perform post-processing, and provide end-user applications with easy access to NetFlow data. If you have enabled NetFlow monitoring of the traffic flowing through your datacenters, this feature enables you to perform the same level of monitoring of the traffic flowing through the Cisco Application Centric Infrastructure (Cisco ACI) fabric.

Instead of hardware directly exporting the records to a collector, the records are processed in the supervisor engine and are exported to standard NetFlow collectors in the required format.

For information about configuring NetFlow with virtual machine networking, see the *Cisco ACI Virtualization Guide*.

Configuring a NetFlow Exporter Policy for VM Networking Using the REST API

The following example XML shows how to configure a NetFlow exporter policy for VM networking using the REST API:

```
<polUni>
    <infraInfra>
        <netflowVmmExporterPol name="vmExporter1" dstAddr="2.2.2.2" dstPort="1234"
        srcAddr="4.4.4.4"/>
    </infraInfra>
</polUni>
```

Configuring NetFlow Infra Selectors Using the REST API

You can use the REST API to configure NetFlow infra selectors. The infra selectors are used for attaching a Netflow monitor to a PHY, port channel, virtual port channel, fabric extender (FEX), or port channel fabric extender (FEXPC) interface.

The following example XML shows how to configure NetFlow infra selectors using the REST API:

```
<infraInfra>
    <!--Create Monitor Policy /-->
    <netflowMonitorPol name='monitor_policy1' descr='This is a monitor policy.'>
        <netflowRsMonitorToRecord tnNetflowRecordPolName='record_policy1' />
    <!-- A Max of 2 exporters allowed per Monitor Policy /-->
```

```

<netflowRsMonitorToExporter tnNetflowExporterPolName='exporter_policy1' />
<netflowRsMonitorToExporter tnNetflowExporterPolName='exporter_policy2' />
</netflowMonitorPol>

<!--Create Record Policy /-->
<netflowRecordPol name='record_policy1' descr='This is a record policy.' match='src-ipv4,src-port' />

<!--Create Exporter Policy /-->
<netflowExporterPol name='exporter_policy1' dstAddr='10.10.1.1' srcAddr='10.10.1.10' ver='v9' descr='This is an exporter policy.'>
    <!--Exporter can be behind app EPG or external L3 EPG (InstP) /-->
    <netflowRsExporterToEPg tDn='uni/tn-t1/ap-app1/epg-epg1' />
    <!--This Ctx needs to be the same Ctx that EPG1's BD is part of /-->
    <netflowRsExporterToCtx tDn='uni/tn-t1/ctx-ctx1' />
</netflowExporterPol>

<!--Node-level Policy for collection Interval /-->
<netflowNodePol name='node_policy1' collectIntvl='500' />

<!-- Node Selectors - usual config /-->
<infraNodeP name="infraNodeP-17" >
    <infraLeafS name="infraLeafS-17" type="range">
        <!-- NOTE: The nodes can also be fex nodes /-->
        <infraNodeBlk name="infraNodeBlk-17" from_="101" to_="101"/>
        <infraRsAccNodePGrp tDn='uni/infra/funcprof/accnodepgroup-nodePGrp1' />
    </infraLeafS>
    <infraRsAccPortP tDn="uni/infra/accportprof-infraAccPortP"/>
</infraNodeP>

<!-- Port Selectors - usual config /-->
<infraAccPortP name="infraAccPortP" >
    <infraHPortsS name="infraHPortsS" type="range">
        <!-- NOTE: The interfaces can also be Port-channels, fex interfaces or fex PCs /-->
        <infraPortBlk name="infraPortBlk" fromCard="1" toCard="1" fromPort="8" toPort="8"/>
        <infraRsAccBaseGrp tDn="uni/infra/funcprof/accportgrp-infraAccPortGrp" />
    </infraHPortsS>
</infraAccPortP>

<!-- Policy Groups - usual config /-->
<infraFuncP>
    <!-- Node Policy Group - to setup Netflow Node Policy /-->
    <infraAccNodePGrp name='nodePGrp1' >
        <infraRsNetflowNodePol tnNetflowNodePolName='node_policy1' />
    </infraAccNodePGrp>

    <!-- Access Port Policy Group - to setup Netflow Monitor Policy /-->
    <infraAccPortGrp name="infraAccPortGrp" >
        <!--One Monitor Policy per address family (ipv4, ipv6, ce) /-->
        <infraRsNetflowMonitorPol tnNetflowMonitorPolName='monitor_policy1' fltType='ipv4' />
        <infraRsNetflowMonitorPol tnNetflowMonitorPolName='monitor_policy2' fltType='ipv6' />
        <infraRsNetflowMonitorPol tnNetflowMonitorPolName='monitor_policy2' fltType='ce' />
    </infraAccPortGrp>
</infraFuncP>
</infraInfra>

```

Configuring the NetFlow Tenant Hierarchy Using the REST API

You can use the REST API to configure the NetFlow tenant hierarchy. The tenant hierarchy is used for attaching a NetFlow monitor to a bridge domain, Layer 3 sub-interface, or Layer 3 switched virtual interface (SVI).

The following example XML shows how to configure the NetFlow tenant hierarchy using the REST API:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- api/policymgr/mo/.xml -->
<polUni>
    <fvTenant name="t1">

        <!--Create Monitor Policy /-->
        <netflowMonitorPol name='monitor_policy1' descr='This is a monitor policy.'>
            <netflowRsMonitorToRecord tnNetflowRecordPolName='record_policy1' />
            <!-- A Max of 2 exporters allowed per Monitor Policy /-->
            <netflowRsMonitorToExporter tnNetflowExporterPolName='exporter_policy1' />
            <netflowRsMonitorToExporter tnNetflowExporterPolName='exporter_policy2' />
        </netflowMonitorPol>
        <!--Create Record Policy /-->
        <netflowRecordPol name='record_policy1' descr='This is a record policy.'/>

        <!--Create Exporter Policy /-->
        <netflowExporterPol name='exporter_policy1' dstAddr='10.0.0.1' srcAddr='10.0.0.4'>
            <!--Exporter can be behind app EPG or external L3 EPG (InstP) /-->
            <netflowRsExporterToEPg tDn='uni/tn-t1/ap-app1/epg-epg2' />
            <!--netflowRsExporterToEPg tDn='uni/tn-t1/out-out1/instP-accountingInst' /-->
            <!--This Ctx needs to be the same Ctx that EPG2's BD is part of /-->
            <netflowRsExporterToCtx tDn='uni/tn-t1/ctx-ctx1' />
        </netflowExporterPol>

        <!--Create 2nd Exporter Policy /-->
        <netflowExporterPol name='exporter_policy2' dstAddr='11.0.0.1' srcAddr='11.0.0.4'>
            <netflowRsExporterToEPg tDn='uni/tn-t1/ap-app1/epg-epg2' />
            <netflowRsExporterToCtx tDn='uni/tn-t1/ctx-ctx1' />
        </netflowExporterPol>

        <fvCtx name="ctx1" />

        <fvBD name="bd1" unkMacUcastAct="proxy" >
            <fvSubnet descr="" ip="11.0.0.0/24" />
            <fvRsCtx tnFvCtxName="ctx1" />

            <!--One Monitor Policy per address family (ipv4, ipv6, ce) /-->
            <fvRsBDToNetflowMonitorPol tnNetflowMonitorPolName='monitor_policy1' fltType='ipv4' />
            <fvRsBDToNetflowMonitorPol tnNetflowMonitorPolName='monitor_policy2' fltType='ipv6' />
            <fvRsBDToNetflowMonitorPol tnNetflowMonitorPolName='monitor_policy2' fltType='ce' />
        </fvBD>

        <!--Create App EPG /-->
        <fvAp name="app1">
            <fvAEPg name="epg2" >
                <fvRsBd tnFvBDName="bd1" />
                <fvRsPathAtt encap="vlan-20" instrImedcy="lazy" mode="regular" tDn="topology/pod-1/paths-101/pathep-[eth1/20]" />
            </fvAEPg>
        </fvAp>

        <!--L3 Netflow Config for sub-intf and SVI /-->
        <l3extOut name="out1">
            <l3extLNodeP name="lnodep1" >
                <l3extRsNodeL3OutAtt tDn="topology/pod-1/node-101" rtrId="1.2.3.4" />
                <l3extLIfP name='lifp1'>
                    <!--One Monitor Policy per address family (ipv4, ipv6, ce) /-->
                    <l3extRsLIfPToNetflowMonitorPol tnNetflowMonitorPolName='monitor_policy1' fltType='ipv4' />
                    <l3extRsLIfPToNetflowMonitorPol tnNetflowMonitorPolName='monitor_policy2' fltType='ipv6' />
                    <l3extRsLIfPToNetflowMonitorPol tnNetflowMonitorPolName='monitor_policy2' fltType='ce' />
                </l3extLIfP>
            </l3extLNodeP>
        </l3extOut>
    </fvTenant>
</polUni>
```

<!--Sub-interface 1/40.40 on node 101 /-->

```

<!--ExtRsPathL3OutAtt tDn="topology/pod-1/paths-101/pathep-[eth1/40]"
ifInstT='sub-interface' encap='vlan-40' />

<!--SVI 50 attached to eth1/25 on node 101 /-->
<!--ExtRsPathL3OutAtt tDn="topology/pod-1/paths-101/pathep-[eth1/25]"
ifInstT='external-svi' encap='vlan-50' />
</l3extLIfP>
</l3extLNodeP>

<!--External L3 EPG for Exporter behind external L3 Network /-->
<l3extInstP name="accountingInst">
    <l3extSubnet ip="11.0.0.0/24" />
</l3extInstP>
<l3extRsEctx tnFvCtxName="ctx1"/>
</l3extOut>
</fvTenant>
</polUni>

```

Consuming a NetFlow Exporter Policy Under a VMM Domain Using the REST API

The following example XML shows how to consume a NetFlow exporter policy under a VMM domain using the REST API:

```

<polUni>
    <vmmProvP vendor="VMware">
        <vmmDomP name="mininet">
            <vmmVSwitchPolicyCont>
                <vmmRsVswitchExporterPol tDn="uni/infra/vmmexporterpol-vmExporter1"
activeFlowTimeOut="62" idleFlowTimeOut="16" samplingRate="1"/>
            </vmmVSwitchPolicyCont>
        </vmmDomP>
    </vmmProvP>
</polUni>

```

Configuring the NetFlow or Tetration Analytics Priority Using the REST API

You can specify whether to use the NetFlow or Cisco Tetration Analytics feature by setting the `FeatureSel` attribute of the `<fabricNodeControl>` element. The `FeatureSel` attribute can have one of the following values:

- `analytics`—Specifies Cisco Tetration Analytics. This is the default value.
- `netflow`—Specifies NetFlow.

The following example REST API post specifies for the switch "test1" to use the NetFlow feature:

```

http://192.168.10.1/api/node/mo/uni/fabric.xml
<fabricNodeControl name="test1" FeatureSel="netflow" />

```

ACL Contract Permit and Deny Logging

About ACL Contract Permit and Deny Logs

To log and/or monitor the traffic flow for a contract rule, you can enable and view the logging of packets or flows that were allowed to be sent because of contract permit rules and the logging of packets or flows that were dropped because of taboo contract deny rules.

Enabling Taboo Contract Deny Logging Using the REST API

The following example shows you how to enable Taboo Contract deny logging using the REST API.

Procedure

To enable ACL deny logging, post data such as the follow example:

```
POST https://192.0.20.123/api/node/mo/uni/tn-sgladwin_t1/tabcoo-TCP_Taboo_Contract.json
{
  "vzTaboo": {
    "attributes": {
      "dn": "uni/tn-sgladwin_t1/tabcoo-TCP_Taboo_Contract",
      "name": "TCP_Taboo_Contract",
      "rn": "tabcoo-TCP_Taboo_Contract",
      "status": "created"
    },
    "children": [
      "vzTSubj": {
        "attributes": {
          "dn": "uni/tn-sgladwin_t1/tabcoo-TCP_Taboo_Contract/tsubj-TCP_Filter_Subject",
          "name": "TCP_Filter_Subject",
          "rn": "tsubj-TCP_Filter_Subject",
          "status": "created"
        },
        "children": [
          "vzRsDenyRule": {
            "attributes": {
              "tnVzFilterName": "TCP_Filter",
              "directives": "log",
              "status": "created"
            },
            "children": []
          }
        ]
      }
    ],
    "response": {"totalCount": "0", "imdata": []}
  }
}
```

DOM Statistics

About Digital Optical Monitoring

Real-time digital optical monitoring (DOM) data is collected from SFPs, SFP+, and XFPs periodically and compared with warning and alarm threshold table values. The DOM data collected are transceiver transmit bias current, transceiver transmit power, transceiver receive power, and transceiver power supply voltage.

Enabling Digital Optical Monitoring Using the REST API

Before you can view digital optical monitoring (DOM) statistics about a physical interface, enable DOM on the interface.

To enable DOM using the REST API:

Procedure

Step 1 Create a fabric node control policy (fabricNodeControlPolicy) as in the following example:

```
<fabricNodeControl dn="uni/fabric/nodecontrol-testdom" name="testdom" control="1"
rn="nodecontrol-testdom" status="created" />
```

Step 2 Associate a fabric node control policy to a policy group as follows:

```
<?xml version="1.0" encoding="UTF-8" ?>
<fabricLeNodePGrp dn="uni/fabric/funcprof/lenodepgrp-nodegrp2" name="nodegrp2"
rn="lenodepgrp-nodegrp2" status="created,modified" >

    <fabricRsMonInstFabricPol tnMonFabricPolName="default" status="created,modified" />
    <fabricRsNodeCtrl tnFabricNodeControlName="testdom" status="created,modified" />

</fabricLeNodePGrp>
```

Step 3 Associate a policy group to a switch (in the following example, the switch is 103) as follows:

```
<?xml version="1.0" encoding="UTF-8" ?>
<fabricLeafP>
    <attributes>
        <dn>uni/fabric/leprof-leafSwitchProfile</dn>
        <name>leafSwitchProfile</name>
        <rn>leprof-leafSwitchProfile</rn>
        <status>created,modified</status>
    </attributes>
    <children>
        <fabricLeafS>
            <attributes>
                <dn>uni/fabric/leprof-leafSwitchProfile/leaves-test-typ-range</dn>
                <type>range</type>
                <name>test</name>
                <rn>leaves-test-typ-range</rn>
                <status>created,modified</status>
            </attributes>
            <children>
                <fabricNodeBlk>
                    <attributes>

<dn>uni/fabric/leprof-leafSwitchProfile/leaves-test-typ-range/nodeblk-09533c1d228097da</dn>

                    <from_>103</from_>
                    <to_>103</to_>
                    <name>09533c1d228097da</name>
                    <rn>nodeblk-09533c1d228097da</rn>
                    <status>created,modified</status>
                </attributes>
            </fabricNodeBlk>
        </children>
        <children>
            <fabricRsLeNodePGrp>
                <attributes>
                    <tDn>uni/fabric/funcprof/lenodepgrp-nodegrp2</tDn>
                    <status>created</status>
                </attributes>
            </fabricRsLeNodePGrp>
        </children>
    </fabricLeafS>
```

```
</children>
</fabricLeafP>
```

Syslog

About Syslog

During operation, a fault or event in the Cisco Application Centric Infrastructure (ACI) system can trigger the sending of a system log (syslog) message to the console, to a local file, and to a logging server on another system. A system log message typically contains a subset of information about the fault or event. A system log message can also contain audit log and session log entries.


Note

For a list of syslog messages that the APIC and the fabric nodes can generate, see http://www.cisco.com/c/en/us/td/docs/switches/datacenter/aci/apic/sw/1-x/syslog/guide/aci_syslog/ACI_SysMsg.html.

Many system log messages are specific to the action that a user is performing or the object that a user is configuring or administering. These messages can be the following:

- Informational messages, providing assistance and tips about the action being performed
- Warning messages, providing information about system errors related to an object, such as a user account or service profile, that the user is configuring or administering

In order to receive and monitor system log messages, you must specify a syslog destination, which can be the console, a local file, or one or more remote hosts running a syslog server. In addition, you can specify the minimum severity level of messages to be displayed on the console or captured by the file or host. The local file for receiving syslog messages is `/var/log/external/messages`.

A syslog source can be any object for which an object monitoring policy can be applied. You can specify the minimum severity level of messages to be sent, the items to be included in the syslog messages, and the syslog destination.

You can change the display format for the Syslogs to NX-OS style format.

Additional details about the faults or events that generate these system messages are described in the *Cisco APIC Faults, Events, and System Messages Management Guide*, and system log messages are listed in the *Cisco ACI System Messages Reference Guide*.


Note

Not all system log messages indicate problems with your system. Some messages are purely informational, while others may help diagnose problems with communications lines, internal hardware, or the system software.

Configuring a Syslog Group and Destination Using the REST API

This procedure configures syslog data destinations for logging and evaluation. You can export syslog data to the console, to a local file, or to one or more syslog servers in a destination group. This example sends alerts to the console, information to a local file, and warnings to a remote syslog server.

Procedure

To create a syslog group and destination using the REST API, send a post with XML such as the following example:

Example:

```
<syslogGroup name="tenant64_SyslogDest" format="aci"
dn="uni/fabric/slgroup-tenant64_SyslogDest">
    <syslogConsole name="" format="aci" severity="alerts" adminState="enabled"/>
    <syslogFile name="" format="aci" severity="information" adminState="enabled"/>
    <syslogProf name="syslog" adminState="enabled"/>
    <syslogRemoteDest name="Syslog_remoteDest" format="aci" severity="warnings"
        adminState="enabled" port="514" host="192.168.100.20" forwardingFacility="local7">
        <fileRsARemoteHostToEpg tDn="uni/tn-mgmt/mgmtp-default/oob-default"/>
    </syslogRemoteDest>
</syslogGroup>
```

Creating a Syslog Source Using the REST API

A syslog source can be any object for which an object monitoring policy can be applied.

Before You Begin

Create a syslog monitoring destination group.

Procedure

To create a syslog source, send a POST request with XML such as the following example:

Example:

```
<syslogSrc
    name="VRF64_SyslogSource" minSev="warnings" incl="faults"
    dn="uni/tn-tenant64/monepg-MonPol1/slsrc-VRF64_SyslogSource">
    <syslogRsDestGroup tDn="uni/fabric/slgroup-tenant64_SyslogDest"/>
</syslogSrc>
```

Enabling Syslog to Display in NX-OS CLI Format, Using the REST API

By default the Syslog format is RFC 5424 compliant. You can change the default display of Syslogs to NX-OS type format, similar to the following example:

```
apic1# moquery -c "syslogRemoteDest"
Total Objects shown: 1

# syslog.RemoteDest
host          : 172.23.49.77
adminState     : enabled
childAction    :
descr         :
```

```

dn                  : uni/fabric/slgroup-syslog-mpod/rdst-172.23.49.77
epgDn              :
format             : nxos
forwardingFacility : local7
ip                 :
lcOwn              : local
modTs              : 2016-05-17T16:51:57.231-07:00
monPolDn           : uni/fabric/monfab-default
name               : syslog-dest
operState          : unknown
port               : 514
rn                 : rdst-172.23.49.77
severity            : information
status              :
uid                : 15374
vrfId              : 0
vrfName             :

```

To enable the Syslogs to display in NX-OS type format, perform the following steps, using the REST API.

Procedure

-
- Step 1** Enable the Syslogs to display in NX-OS type format, as in the following example:

```

POST https://192.168.20.123/api/node/mo/uni/fabric.xml
<syslogGroup name="DestGrp77" format="nxos">
<syslogRemoteDest name="slRmtDest77" host="172.31.138.20" severity="debugging"/>
</syslogGroup>

```

The syslogGroup is the Syslog monitoring destination group, the sysLogRemoteDest is the name you previously configured for your Syslog server, and the host is the IP address for the previously configured Syslog server.

- Step 2** Set the Syslog format back to the default RFC 5424 format, as in the following example:

```

POST https://192.168.20.123/api/node/mo/uni/fabric.xml
<syslogGroup name="DestGrp77" format="aci">
<syslogRemoteDest name="slRmtDest77" host="172.31.138.20" severity="debugging"/>
</syslogGroup>

```

Data Plane Policing

Overview

This article provides an example of how to configure Data Plane Policing.

Use data plane policing (DPP) to manage bandwidth consumption on ACI fabric access interfaces. DPP policies can apply to egress traffic, ingress traffic, or both. DPP monitors the data rates for a particular interface. When the data rate exceeds user-configured values, marking or dropping of packets occurs immediately. Policing does not buffer the traffic; therefore, the transmission delay is not affected. When traffic exceeds the data rate, the ACI fabric can either drop the packets or mark QoS fields in them.

DPP policies can be single-rate, dual-rate, and color-aware. Single-rate policies monitor the committed information rate (CIR) of traffic. Dual-rate policers monitor both CIR and peak information rate (PIR) of traffic. In addition, the system monitors associated burst sizes. Three colors, or conditions, are determined by

the policer for each packet depending on the data rate parameters supplied: conform (green), exceed (yellow), or violate (red).

Typically, DPP policies are applied to physical or virtual layer 2 connections for virtual or physical devices such as servers or hypervisors, and on layer 3 connections for routers. DPP policies applied to leaf switch access ports are configured in the fabric access (infra) portion of the ACI fabric, and must be configured by a fabric administrator. DPP policies applied to interfaces on border leaf switch access ports (l3extOut or l2extOut) are configured in the tenant (fvTenant) portion of the ACI fabric, and can be configured by a tenant administrator.

Only one action can be configured for each condition. For example, a DPP policy can conform to the data rate of 256000 bits per second, with up to 200 millisecond bursts. The system applies the conform action to traffic that falls within this rate, and it would apply the violate action to traffic that exceeds this rate. Color-aware policies assume that traffic has been previously marked with a color. This information is then used in the actions taken by this type of policer.



Note

The following are not supported on all platforms:

- Egress Data plane policers on Switched Virtual Interfaces (except N9K-C93180YC-EX)
- FEX ports

The following are not supported on N9K-C93180YC-EX only:

- 2 Rate and 3 Color policers
- Confirm mark action
- Packet policer

Configuring Data Plane Policing Using the REST API

To police the L2 traffic coming in to the Leaf:

```
<!-- api/node/mo/uni/.xml -->
<infraInfra>
<qosDppPol name="infradpp5" burst="2000" rate="2000" be="400"/>
<!--
    List of nodes. Contains leaf selectors. Each leaf selector contains list of node blocks
-->
<infraNodeP name="leaf1">
<infraLeafS name="leaf1" type="range">
<infraNodeBlk name="leaf1" from_= "101" to_= "101"/>
</infraLeafS>
<infraRsAccPortP tDn="uni/infra/accportprof-portselector1"/>
</infraNodeP>
<!--
    PortP contains port selectors. Each port selector contains list of ports. It
    also has association to port group policies
-->
<infraAccPortP name="portselector1">
<infraHPortS name="pselc" type="range">
<infraPortBlk name="blk" fromCard="1" toCard="1" fromPort="48" toPort="49"></infraPortBlk>
<infraRsAccBaseGrp tDn="uni/infra/funcprof/accportgrp-portSet2"/>
</infraHPortS>
</infraAccPortP>
<!-- FuncP contains access bundle group policies -->
<infraFuncP>
```

```
<infraAccPortGrp name="portSet2">
<infraRsQosIngressDppIfPol tnQosDppPolName="infradpp5"/>
</infraAccPortGrp>
</infraFuncP>
</infraInfra>
```

To police the L2 traffic going out of the Leaf:

```
<!-- api/node/mo/uni/.xml -->
<infraInfra>
<qosDppPol name="infradpp2" burst="4000" rate="4000"/>
<!--
List of nodes. Contains leaf selectors. Each leaf selector contains list of node blocks
-->
<infraNodeP name="leaf1">
<infraLeafS name="leaf1" type="range">
<infraNodeBlk name="leaf1" from_="101" to_="101"/>
</infraLeafS>
<infraRsAccPortP tDn="uni/infra/accportprof-portselector2"/>
</infraNodeP>
<!--
PortP contains port selectors. Each port selector contains list of ports. It
also has association to port group policies
-->
<infraAccPortP name="portselector2">
<infraHPortS name="pselc" type="range">
<infraPortBlk name="blk" fromCard="1" toCard="1" fromPort="37" toPort="38"></infraPortBlk>
<infraRsAccBaseGrp tDn="uni/infra/funcprof/accportgrp-portSet2"/>
</infraHPortS>
</infraAccPortP>
<!-- FuncP contains access bundle group policies -->
<infraFuncP>
<infraAccPortGrp name="portSet2">
<infraRsQosEgressDppIfPol tnQosDppPolName="infradpp2"/>
</infraAccPortGrp>
</infraFuncP>
</infraInfra>
```

To police the L3 traffic coming in to the Leaf:

```
<!-- api/node/mo/uni/.xml -->
<fvTenant name="dppTenant">
<qosDppPol name="gmeo" burst="2000" rate="2000"/>
<l3extOut name="Outside">
<l3extInstP name="extroute"/>
<l3extLNodeP name="borderLeaf">
<l3extRsNodeL3OutAtt tDn="topology/pod-1/node-101" rtrId="10.0.0.1">
<ipRouteP ip="0.0.0.0">
<ipNexthopP nhAddr="192.168.62.2"/>
</ipRouteP>
</l3extRsNodeL3OutAtt>
<l3extLIfP name="portProfile">
<l3extRsPathL3OutAtt addr="192.168.40.1/30" ifInstT="l3-port"
tDn="topology/pod-1/paths-101/pathep-[eth1/40]"/>
<l3extRsPathL3OutAtt addr="192.168.41.1/30" ifInstT="l3-port"
tDn="topology/pod-1/paths-101/pathep-[eth1/41]"/>
<l3extRsIngressQosDppPol tnQosDppPolName="gmeo"/>
</l3extLIfP>
</l3extLNodeP>
</l3extOut>
</fvTenant>
```

To police the L3 traffic going out of the Leaf:

```
<!-- api/node/mo/uni/.xml -->
<fvTenant name="dppTenant">
<qosDppPol name="gmeo" burst="2000" rate="2000"/>
<l3extOut name="Outside">
<l3extInstP name="extroute"/>
<l3extLNodeP name="borderLeaf">
<l3extRsNodeL3OutAtt tDn="topology/pod-1/node-101" rtrId="10.0.0.1">
<ipRouteP ip="0.0.0.0">
<ipNexthopP nhAddr="192.168.62.2"/>
</ipRouteP>
</l3extRsNodeL3OutAtt>
<l3extLIfP name="portProfile">
```

```

<13extRsPathL3OutAtt addr="192.168.40.1/30" ifInstT="13-port"
tDn="topology/pod-1/paths-101/pathep-[eth1/40]"/>
<13extRsPathL3OutAtt addr="192.168.41.1/30" ifInstT="13-port"
tDn="topology/pod-1/paths-101/pathep-[eth1/41]"/>
<13extRsEgressQosDppPol tnQosDppPolName="gmeo"/>
</13extLIfP>
</13extLNodeP>
</13extOut>
</fvTenant>

```

Traffic Storm Control

About Traffic Storm Control

A traffic storm occurs when packets flood the LAN, creating excessive traffic and degrading network performance. You can use traffic storm control policies to prevent disruptions on Layer 2 ports by broadcast, unknown multicast, or unknown unicast traffic storms on physical interfaces.

By default, storm control is not enabled in the ACI fabric. ACI bridge domain (BD) Layer 2 unknown unicast flooding is enabled by default within the BD but can be disabled by an administrator. In that case, a storm control policy only applies to broadcast and unknown multicast traffic. If Layer 2 unknown unicast flooding is enabled in a BD, then a storm control policy applies to Layer 2 unknown unicast flooding in addition to broadcast and unknown multicast traffic.

Traffic storm control (also called traffic suppression) allows you to monitor the levels of incoming broadcast, multicast, and unknown unicast traffic over a one second interval. During this interval, the traffic level, which is expressed either as percentage of the total available bandwidth of the port or as the maximum packets per second allowed on the given port, is compared with the traffic storm control level that you configured. When the ingress traffic reaches the traffic storm control level that is configured on the port, traffic storm control drops the traffic until the interval ends. An administrator can configure a monitoring policy to raise a fault when a storm control threshold is exceeded.

Configuring a Traffic Storm Control Policy Using the REST API

To configure a traffic storm control policy, create a `stormctrl:IfPol` object with the desired properties.

To create a policy named MyStormPolicy, send this HTTP POST message:

```
POST https://192.0.20.123/api/mo/uni/infra/stormctrlifp-MyStormPolicy.json
```

In the body of the POST message, Include the following JSON payload structure to specify the policy by percentage of available bandwidth:

```

{
  "stormctrlIfPol": {
    "attributes": {
      "dn": "uni/infra/stormctrlifp-MyStormPolicy",
      "name": "MyStormPolicy",
      "rate": "75",
      "burstRate": "85",
      "rn": "stormctrlifp-MyStormPolicy",
      "status": "created"
    },
    "children": []
  }
}

```

In the body of the POST message, Include the following JSON payload structure to specify the policy by packets per second:

```
{"stormctrlIfPol":  
  {"attributes":  
    {"dn":"uni/infra/stormctrlifp-MyStormPolicy",  
     "name":"MyStormPolicy",  
     "ratePps":"12000",  
     "burstPps":"15000",  
     "rn":"stormctrlifp-MyStormPolicy",  
     "status":"created"  
    },  
    "children":[]  
  }  
}
```

Apply the traffic storm control interface policy to an interface port.

POST
<http://192.0.20.123/api/node/mo/uni/infra/funcprof/accportgrp-InterfacePolicyGroup/rsstormctrlIfPol.json>
In the body of the POST message, Include the following JSON payload structure to apply the policy to the interface policy group.

```
{"infraRsStormctrlIfPol":{"attributes":{"tnStormctrlIfPolName":"testStormControl"},"children":[]}}
```



CHAPTER 12

Provisioning Layer 2 Networks

- Networking Domains, VLANs, and AEPs, page 177
- Interfaces, page 181
- FCoE, page 191
- 802.1Q Tunnels, page 205
- Breakout Ports, page 210
- IGMP Snooping, page 214
- Proxy ARP, page 218

Networking Domains, VLANs, and AEPs

Networking Domains

A fabric administrator creates domain policies that configure ports, protocols, VLAN pools, and encapsulation. These policies can be used exclusively by a single tenant, or shared. Once a fabric administrator configures domains in the ACI fabric, tenant administrators can associate tenant endpoint groups (EPGs) to domains.

These networking domain profiles can be configured:

- VMM domain profiles (`vmmDomP`) are required for virtual machine hypervisor integration.
- Physical domain profiles (`physDomP`) are typically used for bare metal server attachment and management access.
- Bridged outside network domain profiles (`12extDomP`) are typically used to connect a bridged external network trunk switch to a leaf switch in the ACI fabric.
- Routed outside network domain profiles (`13extDomP`) are used to connect a router to a leaf switch in the ACI fabric.

A domain is configured to be associated with a VLAN pool. EPGs are then configured to use the VLANs associated with a domain.

**Note**

EPG port and VLAN configurations must match those specified in the domain infrastructure configuration with which the EPG associates. If not, the APIC will raise a fault. When such a fault occurs, verify that the domain infrastructure configuration matches the EPG port and VLAN configurations.

Configuring a Physical Domain Using the REST API

A physical domain acts as the link between the VLAN pool and the Access Entity Profile (AEP). The domain also ties the fabric configuration to the tenant configuration, as the tenant administrator is the one who associates domains to EPGs, while the domains are created under the fabric tab. When configuring in this order, only the profile name and the VLAN pool are configured.

Procedure

Configure a physical domain by sending a post with XML such as the following example:

Example:

```
<physDomP dn="uni/phys-bsprint-PHY" lcOwn="local"
modTs="2015-02-23T16:13:21.906-08:00" monPolDn="uni/fabric/monfab-default"
name="bsprint-PHY" ownerKey="" ownerTag="" status="" uid="8131">
<infraRsVlanNs childAction="" forceResolve="no" lcOwn="local" modTs="2015-02-
23T16:13:22.065-08:00" monPolDn="uni/fabric/monfab-default" rType="mo" rn="rsvlanNs"
state="formed" stateQual="none" status="" tCl="fvnsVlanInstP" tDn="uni/infra/vlanns-
[bsprint-vlan-pool]-static" tType="mo" uid="8131"/>
<infraRsVlanNsDef forceResolve="no" lcOwn="local" modTs="2015-02-
23T16:13:22.065-08:00" rType="mo" rn="rsvlanNsDef" state="formed" stateQual="none"
status="" tCl="fvnsAInstP" tDn="uni/infra/vlanns-[bsprint-vlan-pool]-static"
tType="mo"/>
<infraRtDomP lcOwn="local" modTs="2015-02-23T16:13:52.945-08:00"
rn="rtDomP-[uni/infra/attentp-bsprint-AEP]" status="" tCl="infraAttEntityP"
tDn="uni/infra/attentp-bsprint-AEP"/>
</physDomP>
```

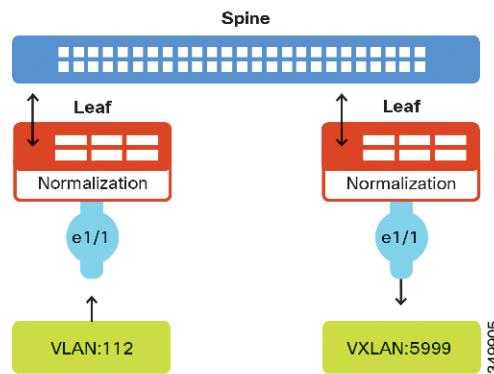
Creating VLAN Pools

In this example, configuring newly-connected bare metal servers first requires creation of a physical domain and then association of the domain to a VLAN pool. As mentioned in the previous section, VLAN pools define a range of VLAN IDs that will be used by the EPGs.

The servers are connected to two different leaf nodes in the fabric. Each server will be tagging using 802.1Q or VXLAN encapsulation. The range of VLANs used in the configuration example is 100-199. As depicted in the following figure, the ACI fabric can also act as a gateway between disparate encapsulation types such as untagged traffic, 802.1Q VLAN tags, VXLAN VNIDs, and NVGRE tags. The leaf switches normalize the traffic by stripping off tags and reapplying the required tags on fabric egress. In ACI, it is important to understand that the definition of VLANs as they pertain to the leaf switch ports is utilized only for identification purposes. When a packet arrives ingress to a leaf switch in the fabric, ACI has to know beforehand how to

classify packets into the different EPGs, using identifiers like VLANs, VXLAN, NVGRE, physical port IDs, virtual port IDs.

Figure 19: Encapsulation normalization



Creating a VLAN Pool Using the REST API

The following example REST request creates a VLAN pool:

```

<fvnsVlanInstP allocMode="static" childAction="" configIssues="" descr=""
  dn="uni/infra/vlanns-[bsprint-vlan-pool]-static" lcOwn="local"
  modTs="2015-02-23T15:58:33.538-08:00"
  monPolDn="uni/fabric/monfab-default" name="bsprint-vlan-pool"
  ownerKey="" ownerTag="" status="" uid="8131">
  <fvnsRtVlanNs childAction="" lcOwn="local" modTs="2015-02-25T11:35:33.365-08:00"
    rn="rtinfraVlanNs-[uni/l2dom-JC-L2-Domain]" status="" tCl="l2extDomP"
    tDn="uni/l2dom-JC-L2-Domain"/>
  <fvnsRtVlanNs childAction="" lcOwn="local" modTs="2015-02-23T16:13:22.007-08:00"
    rn="rtinfraVlanNs-[uni/phys-bsprint-PHY]" status="" tCl="physDomP"
    tDn="uni/physbsprint-PHY"/>
  <fvnsEncapBlk childAction="" descr="" from="vlan-100" lcOwn="local"
  modTs="2015-02-23T15:58:33.538-08:00"
  name="" rn="from-[vlan-100]-to-[vlan-199]" status="" to="vlan-199" uid="8131"/>
</fvnsVlanInstP>
  
```

Attachable Entity Profile

The ACI fabric provides multiple attachment points that connect through leaf ports to various external entities such as bare metal servers, virtual machine hypervisors, Layer 2 switches (for example, the Cisco UCS fabric interconnect), or Layer 3 routers (for example Cisco Nexus 7000 Series switches). These attachment points can be physical ports, FEX ports, port channels, or a virtual port channel (vPC) on leaf switches.

**Note**

When creating a VPC domain between two leaf switches, both switches must be in the same switch generation, one of the following:

- Generation 1 - Cisco Nexus N9K switches without “EX” on the end of the switch name; for example, N9K-9312TX
- Generation 2 – Cisco Nexus N9K switches with “EX” on the end of the switch model name; for example, N9K-93108TC-EX

Switches such as these two are not compatible VPC peers. Instead, use switches of the same generation

An Attachable Entity Profile (AEP) represents a group of external entities with similar infrastructure policy requirements. The infrastructure policies consist of physical interface policies that configure various protocol options, such as Cisco Discovery Protocol (CDP), Link Layer Discovery Protocol (LLDP), or Link Aggregation Control Protocol (LACP).

An AEP is required to deploy VLAN pools on leaf switches. Encapsulation blocks (and associated VLANs) are reusable across leaf switches. An AEP implicitly provides the scope of the VLAN pool to the physical infrastructure.

The following AEP requirements and dependencies must be accounted for in various configuration scenarios, including network connectivity, VMM domains, and multipod configuration:

- The AEP defines the range of allowed VLANs but it does not provision them. No traffic flows unless an EPG is deployed on the port. Without defining a VLAN pool in an AEP, a VLAN is not enabled on the leaf port even if an EPG is provisioned.
- A particular VLAN is provisioned or enabled on the leaf port that is based on EPG events either statically binding on a leaf port or based on VM events from external controllers such as VMware vCenter or Microsoft Azure Service Center Virtual Machine Manager (SCVMM).
- Attached entity profiles can be associated directly with application EPGs, which deploy the associated application EPGs to all those ports associated with the attached entity profile. The AEP has a configurable generic function (infraGeneric), which contains a relation to an EPG (infraRsFuncToEpg) that is deployed on all interfaces that are part of the selectors that are associated with the attachable entity profile.

A virtual machine manager (VMM) domain automatically derives physical interface policies from the interface policy groups of an AEP.

An override policy at the AEP can be used to specify a different physical interface policy for a VMM domain. This policy is useful in scenarios where a VM controller is connected to the leaf switch through an intermediate Layer 2 node, and a different policy is desired at the leaf switch and VM controller physical ports. For example, you can configure LACP between a leaf switch and a Layer 2 node. At the same time, you can disable LACP between the VM controller and the Layer 2 switch by disabling LACP under the AEP override policy.

Creating an Attachable Access Entity Profile Using the REST API

The following example REST request creates an attachable access entity profile (AEP):

```
<infraAttEntityP childAction="" configIssues="" descr="" dn="uni/infra/attentpbsprint-AEP"
  lcOwn="local" modTs="2015-02-23T16:13:52.874-08:00" monPolDn="uni/fabric/monfab-default"
  name="bsprint-AEP" ownerKey="" ownerTag="" status="" uid="8131">
  <infraContDomP childAction="" lcOwn="local" modTs="2015-02-23T16:13:52.874-08:00"
    rn="dompcont" status="">
    <infraAssocDomP childAction="" dompDn="uni/phys-bsprint-PHY" lcOwn="local"
      modTs="2015-02-23T16:13:52.961-08:00" rn="assocdomp-[uni/phys-bsprint-PHY]">
```

```

status="" />
<infraAssocDomP childAction="" dompDn="uni/l2dom-JC-L2-Domain" lcOwn="local"
    modTs="2015-02-25T11:35:33.570-08:00" rn="assocdomp-[uni/l2dom-JC-L2-Domain]"
    status="" />
</infraContDomP>
<infraContNS childAction="" lcOwn="local" modTs="2015-02-23T16:13:52.874-08:00"
    monPolDn="uni/fabric/monfab-default" rn="nscont" status="" />
<infraRsToEncapInstDef childAction="" depSt="" forceResolve="no" lcOwn="local"
    modTs="2015-02-23T16:13:52.961-08:00" monPolDn="uni/fabric/monfabdefault"
    rType="mo" rn="rstoEncapInstDef-[allocencap-[uni/infra]/encapsdef-
        [uni/infra/vlanns-[bsprint-vlan-pool]-static]]" state="formed" stateQual="none"
    status="" tCl="stpEncapInstDef" tDn="alloccap-[uni/infra]/encapsdef-
        [uni/infra/vlanns-[bsprint-vlan-pool]-static]" tType="mo">
        <fabricCreatedBy childAction="" creatorDn="uni/l2dom-JC-L2-Domain"
            depSt="" domainDn="uni/l2dom-JC-L2-Domain" lcOwn="local" modTs="2015-02-
                25T11:35:33.570-08:00" monPolDn="uni/fabric/monfab-default" profileDn=""
            rn="source-[uni/l2dom-JC-L2-Domain]" status="" />
        <fabricCreatedBy childAction="" creatorDn="uni/phys-bsprint-PHY" depSt=""
            domainDn="uni/phys-bsprint-PHY" lcOwn="local"
            modTs="2015-02-23T16:13:52.961-08:00"
            monPolDn="uni/fabric/monfab-default" profileDn=""
            rn="source-[uni/phys-bsprint-PHY]" status="" />
        </infraRsToEncapInstDef>
    </infraContNS>
<infraRtAttEntP childAction="" lcOwn="local" modTs="2015-02-24T11:59:37.980-08:00"
    rn="rtattEntP-[uni/infra/funcprof/accportgrp-bsprint-AccessPort]" status=""
    tCl="infraAccPortGrp" tDn="uni/infra/funcprof/accportgrp-bsprint-AccessPort"/>
<infraRsDomP childAction="" forceResolve="no" lcOwn="local" modTs="2015-02-
    25T11:35:33.570-08:00" monPolDn="uni/fabric/monfab-default" rType="mo"
    rn="rsdomP-[uni/l2dom-JC-L2-Domain]" state="formed" stateQual="none" status=""
    tCl="l2extDomP" tDn="uni/l2dom-JC-L2-Domain" tType="mo" uid="8754"/>
<infraRsDomP childAction="" forceResolve="no" lcOwn="local"
    modTs="2015-02-23T16:13:52.961-08:00" monPolDn="uni/fabric/monfab-default" rType="mo"
    rn="rsdomP-[uni/phys-bsprint-PHY]" state="formed" stateQual="none" status=""
    tCl="physDomP"
    tDn="uni/phys-bsprint-PHY" tType="mo" uid="8131"/>
</infraAttEntityP>

```

Interfaces

Ports, PCs, and VPCs

Configuring a Single Port Channel Applied to Multiple Switches

This example creates a port channel on leaf switch 17, another port channel on leaf switch 18, and a third one on leaf switch 20. On each leaf switch, the same interfaces will be part of the port channel (interfaces 1/10 to 1/15 and 1/20 to 1/25). All these port channels will have the same configuration.

Before You Begin

- The ACI fabric is installed, APIC controllers are online, and the APIC cluster is formed and healthy.
- An APIC fabric administrator account is available that will enable creating the necessary fabric infrastructure configurations.
- The target leaf switch and protocol(s) are configured and available.

Procedure

To create the port channel, send a post with XML such as the following:

Example:

```
<infraInfra dn="uni/infra">

    <infraNodeP name="test">
        <infraLeafS name="leafs" type="range">
            <infraNodeBlk name="nblk" from_="17" to_="18"/>
            <infraNodeBlk name="nblk" from_="20" to_="20"/>
        </infraLeafS>
        <infraRsAccPortP tDn="uni/infra/accportprof-test"/>
    </infraNodeP>

    <infraAccPortP name="test">
        <infraHPorts name="pselc" type="range">
            <infraPortBlk name="blk1"
                fromCard="1" toCard="1"
                fromPort="10" toPort="15"/>
            <infraPortBlk name="blk2"
                fromCard="1" toCard="1"
                fromPort="20" toPort="25"/>
            <infraRsAccBaseGrp
                tDn="uni/infra/funcprof/accbundle-bndlgrp"/>
        </infraHPorts>
    </infraAccPortP>

    <infraFuncP>
        <infraAccBndlGrp name="bndlgrp" lagT="link">
            <infraRshIfPol tnFabricHifPolName="default"/>
            <infraRscdpIfPol tnCdpIfPolName="default"/>
            <infraRsLacpPol tnLacpLagPolName="default"/>
        </infraAccBndlGrp>
    </infraFuncP>
</infraInfra>
```

Configuring a Single Virtual Port Channel Across Two Switches Using the REST API

The two steps for creating a virtual port channel across two switches are as follows:

- Create a `fabricExplicitGEp`: this policy specifies the leaf switch that pairs to form the virtual port channel.
- Use the infra selector to specify the interface configuration.

The APIC performs several validations of the `fabricExplicitGEp` and faults are raised when any of these validations fail. A leaf can be paired with only one other leaf. The APIC rejects any configuration that breaks this rule. When creating a `fabricExplicitGEp`, an administrator must provide the IDs of both of the leaf switches to be paired. The APIC rejects any configuration which breaks this rule. Both switches must be up when `fabricExplicitGEp` is created. If one switch is not up, the APIC accepts the configuration but raises a fault. Both switches must be leaf switches. If one or both switch IDs corresponds to a spine, the APIC accepts the configuration but raises a fault.

Before You Begin

- The ACI fabric is installed, APIC controllers are online, and the APIC cluster is formed and healthy.

- An APIC fabric administrator account is available that will enable creating the necessary fabric infrastructure configurations.
- The target leaf switch and protocol(s) are configured and available.

Procedure

To create the `fabricExplicitGEp` policy and use the intra selector to specify the interface, send a post with XML such as the following example:

Example:

```
<fabricProtPol pairT="explicit">
<fabricExplicitGEp name="tG" id="2">
    <fabricNodePEP id="18"/>
    <fabricNodePEP id="25"/>
    </fabricExplicitGEp>
</fabricProtPol>
```

Configuring Two Port Channels Applied to Multiple Switches Using the REST API

This example creates two port channels (PCs) on leaf switch 17, another port channel on leaf switch 18, and a third one on leaf switch 20. On each leaf switch, the same interfaces will be part of the PC (interface 1/10 to 1/15 for port channel 1 and 1/20 to 1/25 for port channel 2). The policy uses two switch blocks because each a switch block can contain only one group of consecutive switch IDs. All these PCs will have the same configuration.



Note

Even though the PC configurations are the same, this example uses two different interface policy groups. Each Interface Policy Group represents a PC on a switch. All interfaces associated with a given interface policy group are part of the same PCs.

Before You Begin

- The ACI fabric is installed, APIC controllers are online, and the APIC cluster is formed and healthy.
- An APIC fabric administrator account is available that will enable creating the necessary fabric infrastructure configurations.
- The target leaf switch and protocol(s) are configured and available.

Procedure

To create the two PCs, send a post with XML such as the following:

Example:

```
<infraInfra dn="uni/infra">
    <infraNodeP name="test">
        <infraLeafs name="leafs" type="range">
            <infraNodeBlk name="nblk"
                from_="17" to_="18"/>
            <infraNodeBlk name="nblk"
                from_="20" to_="20"/>
        </infraLeafs>
```

```

<infraRsAccPortP tDn="uni/infra/accportprof-test1"/>
<infraRsAccPortP tDn="uni/infra/accportprof-test2"/>
</infraNodeP>

<infraAccPortP name="test1">
    <infraHPorts name="pselc" type="range">
        <infraPortBlk name="blk1"
            fromCard="1" toCard="1"
            fromPort="10" toPort="15"/>
        <infraRsAccBaseGrp
            tDn="uni/infra/funcprof/accbundle-bndlgrp1"/>
    </infraHPorts>
</infraAccPortP>

<infraAccPortP name="test2">
    <infraHPorts name="pselc" type="range">
        <infraPortBlk name="blk1"
            fromCard="1" toCard="1"
            fromPort="20" toPort="25"/>
        <infraRsAccBaseGrp
            tDn="uni/infra/funcprof/accbundle-bndlgrp2" />
    </infraHPorts>
</infraAccPortP>

<infraFuncP>
    <infraAccBndlGrp name="bndlgrp1" lagT="link">
        <infraRsHIfPol tnFabricHIfPolName="default"/>
        <infraRsCdpIfPol tnCdpIfPolName="default"/>
        <infraRsLacpPol tnLacpLagPolName="default"/>
    </infraAccBndlGrp>
    <infraAccBndlGrp name="bndlgrp2" lagT="link">
        <infraRsHIfPol tnFabricHIfPolName="default"/>
        <infraRsCdpIfPol tnCdpIfPolName="default"/>
        <infraRsLacpPol tnLacpLagPolName="default"/>
    </infraAccBndlGrp>
</infraFuncP>

</infraInfra>

```

Configuring a Virtual Port Channel on Selected Port Blocks of Two Switches Using the REST API

This policy creates a single virtual port channel (VPC) on leaf switches 18 and 25, using interfaces 1/10 to 1/15 on leaf 18, and interfaces 1/20 to 1/25 on leaf 25.

Before You Begin

- The ACI fabric is installed, APIC controllers are online, and the APIC cluster is formed and healthy.
- An APIC fabric administrator account is available that will enable creating the necessary fabric infrastructure configurations.
- The target leaf switch and protocol(s) are configured and available.

**Note**

When creating a VPC domain between two leaf switches, both switches must be in the same switch generation, one of the following:

- Generation 1 - Cisco Nexus N9K switches without “EX” on the end of the switch name; for example, N9K-9312TX
- Generation 2 – Cisco Nexus N9K switches with “EX” on the end of the switch model name; for example, N9K-93108TC-EX

Switches such as these two are not compatible VPC peers. Instead, use switches of the same generation.

Procedure

To create the VPC send a post with XML such as the following example:

Example:

```
<infraInfra dn="uni/infra">

    <infraNodeP name="test1">
        <infraLeafS name="leafs" type="range">
            <infraNodeBlk name="nblk1"
                from_="18" to_="18"/>
        </infraLeafS>
        <infraRsAccPortP tDn="uni/infra/accportprof-test1"/>
    </infraNodeP>

    <infraNodeP name="test2">
        <infraLeafS name="leafs" type="range">
            <infraNodeBlk name="nblk1"
                from_="25" to_="25"/>
        </infraLeafS>
        <infraRsAccPortP tDn="uni/infra/accportprof-test2"/>
    </infraNodeP>

    <infraAccPortP name="test1">
        <infraHPorts name="pselc" type="range">
            <infraPortBlk name="blk1"
                fromCard="1" toCard="1"
                fromPort="10" toPort="15"/>
            <infraRsAccBaseGrp
                tDn="uni/infra/funcprof/accbundle-bndlgrp" />
        </infraHPorts>
    </infraAccPortP>

    <infraAccPortP name="test2">
        <infraHPorts name="pselc" type="range">
            <infraPortBlk name="blk1"
                fromCard="1" toCard="1"
                fromPort="20" toPort="25"/>
            <infraRsAccBaseGrp
                tDn="uni/infra/funcprof/accbundle-bndlgrp" />
        </infraHPorts>
    </infraAccPortP>

    <infraFuncP>
        <infraAccBndlGrp name="bndlgrp" lagT="node">
            <infraRsHIfPol tnFabricHIfPolName="default"/>
            <infraRsCdpIfPol tnCdpIfPolName="default"/>
            <infraRsLacpPol tnLacpLagPolName="default"/>
        </infraAccBndlGrp>
    </infraFuncP>
```

```
</infraInfra>
```

Configuring a Virtual Port Channel and Applying it to a Static Port Using the REST API

Before You Begin

- Install the APIC, verify that the APIC controllers are online, and that the APIC cluster is formed and healthy.
- Verify that an APIC fabric administrator account is available that will enable you to create the necessary fabric infrastructure.
- Verify that the target leaf switches are registered in the ACI fabric and available.

Procedure

- Step 1** To build vPCs, send a post with XML such as the following example:

Example:

```
https://apic-ip-address/api/policymgr/mo/.xml
<polUni>
<infraInfra>
<infraNodeP name="switchProfileforVPC_201">
<infraLeafS name="switchProfileforVPC_201" type="range">
<infraNodeBlk name="nodeBlk" from_="201" to_="201"/>
</infraLeafS>
<infraRsAccPortP tDn="uni/infra/accportprof-intProfileforVPC_201"/>
</infraNodeP>
<infraNodeP name="switchProfileforVPC_202">
<infraLeafS name="switchProfileforVPC_202" type="range">
<infraNodeBlk name="nodeBlk" from_="202" to_="202"/>
</infraLeafS>
<infraRsAccPortP tDn="uni/infra/accportprof-intProfileforVPC_202"/>
</infraNodeP>
<infraAccPortP name="intProfileforVPC_201">
<infraHPortS name="vpc201-202" type="range">
<infraPortBlk name="vpcPort1-15" fromCard="1" toCard="1" fromPort="15"
toPort="15"/>
<infraRsAccBaseGrp tDn="uni/infra/funcprof/accbundle-intPolicyGroupforVPC"/>
</infraHPortS>
</infraAccPortP>
<infraAccPortP name="intProfileforVPC_202">
<infraHPortS name="vpc201-202" type="range">
<infraPortBlk name="vpcPort1-1" fromCard="1" toCard="1" fromPort="1"
toPort="1"/>
<infraRsAccBaseGrp tDn="uni/infra/funcprof/accbundle-intPolicyGroupforVPC"/>
</infraHPortS>
</infraAccPortP>
<infraFuncP>
<infraAccBndlGrp name="intPolicyGroupforVPC" lagT="node">
<infraRsAttEntP tDn="uni/infra/attentp-AttEntityProfileforCisco"/>
<infraRscDpIfPol tnCdpIfPolName="CDP_ON" />
<infraRsLacpPol tnLacpLagPolName="IACP_ACTIVE" />
<infraRshIfPol tnFabricHIfPolName="10GigAuto" />
</infraAccBndlGrp>
</infraFuncP>
</infraInfra>
</polUni>
```

- Step 2** To attach the VPC to static port bindings, send a post with XML such as the following:

Example:

```
https://apic-ip-address/api/node/mo/uni.xml
<polUni>
<fvTenant dn="uni/tn-Cisco" name="Cisco" ownerKey="" ownerTag="">
<fvAp name="CCO" ownerKey="" ownerTag="" prio="unspecified">
<fvAEPg matchT="AtleastOne" name="Web" prio="unspecified">
<fvRsPathAtt encap="vlan-1201" instrImedcy="immediate" mode="native"
tDn="topology/pod-1/protpaths-201-202/pathep-[vpc201-202]" />
</fvAEPg>
<fvAEPg matchT="AtleastOne" name="App" prio="unspecified">
<fvRsPathAtt encap="vlan-1202" instrImedcy="immediate" mode="native"
tDn="topology/pod-1/protpaths-201-202/pathep-[vpc201-202]" />
</fvAEPg>
</fvAp>
</fvTenant>
</polUni>
```

Interface Speed

Interface Configuration Guidelines

When configuring interfaces in an ACI fabric, follow these guidelines.

Half duplex at 100Mbps speed is not supported

In an ACI leaf switch that supports 100Mbps speed, the 100Mbps speed is supported only if the link is in full duplex mode and if autonegotiation is configured the same on both the local and remote peer. The ACI leaf switch and the remote link should both be configured in full duplex mode with autonegotiation disabled on both devices or enabled on both devices.

Connecting an SFP module requires a link speed policy

When you connect an SFP module to a new card, you must create a link speed policy for the module to communicate with the card. Follow these steps to create a link speed policy.

- 1 Create an interface policy to specify the link speed, as in this example:

```
<fabricHIfPol name="mySpeedPol" speed="1G"/>
```

- 2 Reference the link speed policy within an interface policy group, as in this example:

```
<infraAccPortGrp name="myGroup">
  <infraRsHIfPol tnFabricHIfPolName="SpeedPol"/>
</infraAccPortGrp>
```

MAC Pinning

MAC pinning is used for pinning VM traffic in a round-robin fashion to each uplink based on the MAC address of the VM. In a normal virtual port channel (vPC), a hash algorithm uses the source and destination MAC address to determine which uplink will carry a packet. In a vPC with MAC pinning, VM1 might be pinned to the first uplink, VM2 pinned to the second uplink, and so on.

MAC pinning is the recommended option for channeling when connecting to upstream switches that do not support Multichassis EtherChannel (MEC).

Consider these guidelines and restrictions when configuring MAC pinning:

- When a host is connected to two leaf switches using a vPC with MAC pinning, reloading one of the two leaf switches can result in a few minutes of traffic disruption.
- In the API, MAC pinning is selected in the LACP policy by setting lacp:LagPol:mode to mac-pin. When the policy is applied to a vPC, the vPC status as shown in pc:AggrIf:pcMode and in pc:AggrIf:operChannelMode is displayed as active, not as mac-pin.

Changing Interface Speed

This task creates a policy that configures the speed for a set of interfaces.

Procedure

To set the speed for a set of interfaces, send a post with XML such as the following example:

Example:

```
<infraInfra dn="uni/infra">

    <infraNodeP name="test1">
        <infraLeafS name="leafs" type="range">
            <infraNodeBlk name="nblk" from_="18" to_="18"/>
        </infraLeafS>
        <infraRsAccPortP tDn="uni/infra/accportprof-test1"/>
    </infraNodeP>

    <infraNodeP name="test2">
        <infraLeafS name="leafs" type="range">
            <infraNodeBlk name="nblk" from_="25" to_="25"/>
        </infraLeafS>
        <infraRsAccPortP tDn="uni/infra/accportprof-test2"/>
    </infraNodeP>

    <infraAccPortP name="test1">
        <infraHPortS name="pselc" type="range">
            <infraPortBlk name="blk1"
                fromCard="1" toCard="1"
                fromPort="10" toPort="15"/>
            <infraRsAccBaseGrp
                tDn="uni/infra/funcprof/accbundle-bndlgrp" />
        </infraHPortS>
    </infraAccPortP>

    <infraAccPortP name="test2">
        <infraHPortS name="pselc" type="range">
            <infraPortBlk name="blk1"
                fromCard="1" toCard="1"
                fromPort="20" toPort="25"/>
            <infraRsAccBaseGrp
                tDn="uni/infra/funcprof/accbundle-bndlgrp" />
        </infraHPortS>
    </infraAccPortP>

    <infraFuncP>
        <infraAccBndlGrp name="bndlgrp" lagT="node">
            <infraRsHIfPol tnFabricHIfPolName="default"/>
            <infraRsCdpIfPol tnCdpIfPolName="default"/>
            <infraRsLacpPol tnLacpLagPolName="default"/>
        </infraAccBndlGrp>
    </infraFuncP>
</infraInfra>
```

```
</infraFuncP>
</infraInfra>
```

FEXs

ACI FEX Guidelines

Observe the following guidelines when deploying a FEX:

- Assuming that no leaf switch front panel ports are configured to deploy EPG and VLANs, a maximum of 10,000 port EPGs are supported for being deployed using a FEX.
- For each FEX port or vPC that includes FEX ports as members, a maximum of 20 EPGs per VLAN are supported.

Configuring an FEX VPC Policy Using the REST API

This task creates a FEX virtual port channel (VPC) policy.

Before You Begin

- The ACI fabric is installed, APIC controllers are online, and the APIC cluster is formed and healthy.
- An APIC fabric administrator account is available that will enable creating the necessary fabric infrastructure configurations.
- The target leaf switch, interfaces, and protocol(s) are configured and available.
- The FEXes are configured, powered on, and connected to the target leaf interfaces



- Note** When creating a VPC domain between two leaf switches, both switches must be in the same switch generation, one of the following:
- Generation 1 - Cisco Nexus N9K switches without "EX" on the end of the switch name; for example, N9K-9312TX
 - Generation 2 – Cisco Nexus N9K switches with "EX" on the end of the switch model name; for example, N9K-93108TC-EX

Switches such as these two are not compatible VPC peers. Instead, use switches of the same generation.

Procedure

To create the policy linking the FEX through a VPC to two switches, send a post with XML such as the following example:

Example:

```
<polUni>
<infraInfra dn="uni/infra">
<infraNodeP name="fexNodeP105">
```

```

<infraLeafS name="leafs" type="range">
    <infraNodeBlk name="test" from_="105" to_="105"/>
</infraLeafS>
<infraRsAccPortP tDn="uni/infra/accportprof-fex116nif105" />
</infraNodeP>

<infraNodeP name="fexNodeP101">
    <infraLeafS name="leafs" type="range">
        <infraNodeBlk name="test" from_="101" to_="101"/>
    </infraLeafS>
    <infraRsAccPortP tDn="uni/infra/accportprof-fex113nif101" />
</infraNodeP>

<infraAccPortP name="fex116nif105">
    <infraHPortS name="pselc" type="range">
        <infraPortBlk name="blk1"
            fromCard="1" toCard="1" fromPort="45" toPort="48" >
        </infraPortBlk>
        <infraRsAccBaseGrp tDn="uni/infra/fexprof-fexHIF116/fexbundle-fex116" fexId="116" />
    </infraHPortS>
</infraAccPortP>

<infraAccPortP name="fex113nif101">
    <infraHPortS name="pselc" type="range">
        <infraPortBlk name="blk1"
            fromCard="1" toCard="1" fromPort="45" toPort="48" >
        </infraPortBlk>
        <infraRsAccBaseGrp tDn="uni/infra/fexprof-fexHIF113/fexbundle-fex113" fexId="113" />
    </infraHPortS>
</infraAccPortP>

<infraFexP name="fexHIF113">
    <infraFexBndlGrp name="fex113"/>
    <infraHPortS name="pselc-fexPC" type="range">
        <infraPortBlk name="blk"
            fromCard="1" toCard="1" fromPort="15" toPort="16" >
        </infraPortBlk>
        <infraRsAccBaseGrp tDn="uni/infra/funcprof/accbundle-fexPCbundle" />
    </infraHPortS>
    <infraHPortS name="pselc-fexVPC" type="range">
        <infraPortBlk name="blk"
            fromCard="1" toCard="1" fromPort="1" toPort="8" >
        </infraPortBlk>
        <infraRsAccBaseGrp tDn="uni/infra/funcprof/accbundle-fexvpbundle" />
    </infraHPortS>
    <infraHPortS name="pselc-fexaccess" type="range">
        <infraPortBlk name="blk"
            fromCard="1" toCard="1" fromPort="47" toPort="47">
        </infraPortBlk>
        <infraRsAccBaseGrp tDn="uni/infra/funcprof/accportgrp-fexaccport" />
    </infraHPortS>
</infraFexP>

<infraFexP name="fexHIF116">
    <infraFexBndlGrp name="fex116"/>
    <infraHPortS name="pselc-fexPC" type="range">
        <infraPortBlk name="blk"
            fromCard="1" toCard="1" fromPort="17" toPort="18" >
        </infraPortBlk>
        <infraRsAccBaseGrp tDn="uni/infra/funcprof/accbundle-fexPCbundle" />
    </infraHPortS>
    <infraHPortS name="pselc-fexVPC" type="range">
        <infraPortBlk name="blk"
            fromCard="1" toCard="1" fromPort="1" toPort="8" >
        </infraPortBlk>
        <infraRsAccBaseGrp tDn="uni/infra/funcprof/accbundle-fexvpbundle" />
    </infraHPortS>
    <infraHPortS name="pselc-fexaccess" type="range">
        <infraPortBlk name="blk"
            fromCard="1" toCard="1" fromPort="47" toPort="47">
        </infraPortBlk>
</infraFexP>
```

```

<infraRsAccBaseGrp tDn="uni/infra/funcprof/accportgrp-fexaccport" />
</infraHPorts>

</infraFexP>

<infraFuncP>
<infraAccBndlGrp name="fexPCbundle" lagT="link">
    <infraRsLacpPol tnLacpLagPolName='staticLag' />
    <infraRsHIfPol tnFabricHIfPolName="1GHIfPol" />
    <infraRsAttEntP tDn="uni/infra/attentp-fexvpcAttEP"/>
</infraAccBndlGrp>

<infraAccBndlGrp name="fexvpcbundle" lagT="node">
    <infraRsLacpPol tnLacpLagPolName='staticLag' />
    <infraRsHIfPol tnFabricHIfPolName="1GHIfPol" />
    <infraRsAttEntP tDn="uni/infra/attentp-fexvpcAttEP"/>
</infraAccBndlGrp>
</infraFuncP>

<fabricHIfPol name="1GHIfPol" speed="1G" />
<infraAttEntityP name="fexvpcAttEP">
    <infraProvAcc name="provfunc"/>
    <infraRsDomP tDn="uni/phys-fexvpcDOM"/>
</infraAttEntityP>

<lacpLagPol dn="uni/infra/lacplagg-staticLag"
    ctrl="suspi-individual,graceful-conv"
    minLinks="2"
    maxLinks="16">
</lacpLagPol>

```

FCoE

Supporting Fibre Channel over Ethernet Traffic on the ACI Fabric

ACI enables you to configure and manage support for Fibre Channel over Ethernet (FCoE) traffic on the ACI fabric.

FCoE is a protocol that encapsulates Fibre Channel (FC) packets within Ethernet packets, thus enabling storage traffic to move seamlessly from a Fibre Channel SAN to an Ethernet network.

A typical implementation of FCoE protocol support on the ACI fabric enables hosts located on the Ethernet-based ACI fabric to communicate with SAN storage devices located on an FC network. The hosts are connecting through virtual F ports deployed on an ACI leaf switch. The SAN storage devices and FC network are connected through an FCF bridge to the ACI fabric through a virtual NP port, deployed on the same ACI leaf switch as is the virtual F port. Virtual NP ports and virtual F ports are also referred to generically as virtual Fibre Channel (vFC) ports.

**Note**

As of release version 2.0(1), FCoE support is limited to 9300-EX hardware.

With release version 2.2(x), the N9K-CP3180LC-EX 40 Gigabit Ethernet (GE) ports can be used as F or NP ports. However, If they are enabled for FCoE, they cannot be enabled for 40GE port breakout. Breakout is not supported with FCoE.

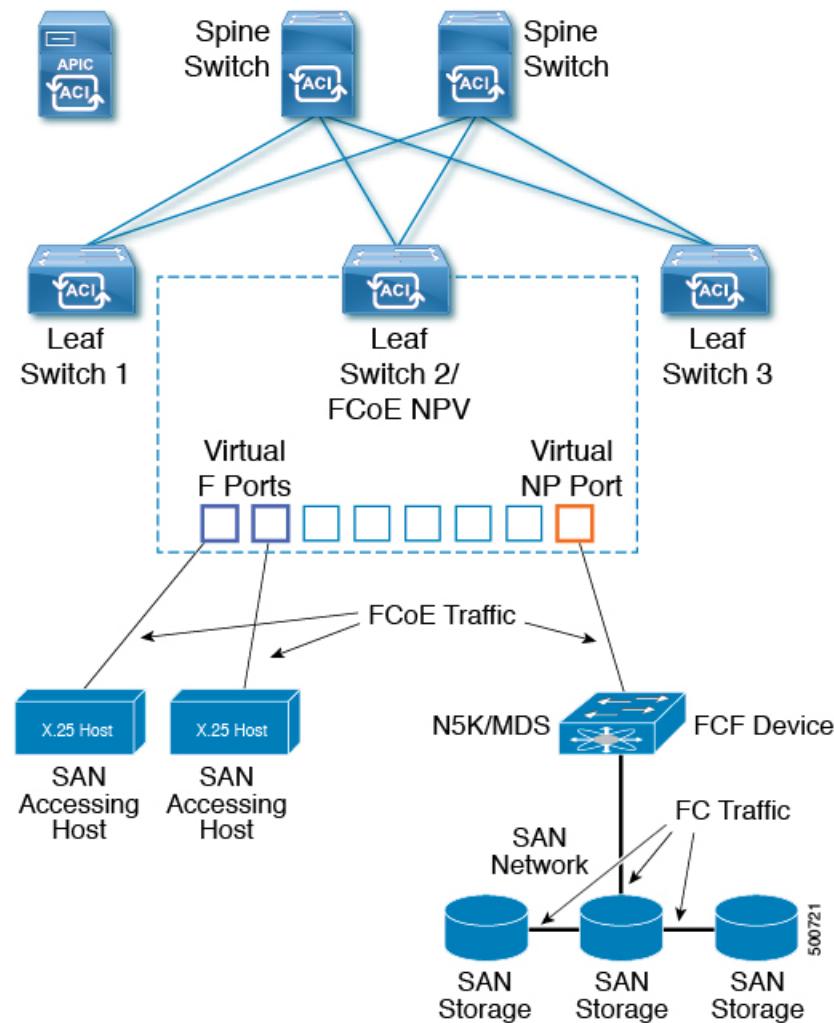
As of release version 2.2(x), FCoE is also supported on the following FEX Nexus devices:

- N2K-C2348UPQ-10GE
 - N2K-C2348TQ-10GE
 - N2K-C2248PQ-10GE
 - B22 FEX for Vendor Blade Servers
-

Topology Supporting FCoE Traffic Through ACI

The topology of a typical configuration supporting FCoE traffic over the ACI fabric consists of the following components:

Figure 20: ACI Topology Supporting FCoE Traffic



- One or more ACI leaf switches configured through FC SAN policies to function as an NPV backbone
- Selected interfaces on the NPV-configured leaf switches configured to function as F ports.
F ports accommodate FCoE traffic to and from hosts running SAN management or SAN-consuming applications.
- Selected interfaces on the NPV-configured leaf switches to function as NP ports.
NP ports accommodate FCoE traffic to and from an FCF bridge.

The FCF bridge receives FC traffic from fibre channel links typically connecting SAN storage devices and encapsulates the FC packets into FCoE frames for transmission over the ACI fabric to the SAN management

or SAN Data-consuming hosts. It receives FCoE traffic and repackages it back to FC for transmission over the fibre channel network.


Note

In the above ACI topology, FCoE traffic support requires direct connections between the hosts and F ports and direct connections between the FCF device and NP port.

APIC servers enable an operator to configure and monitor the FCoE traffic through the APIC Basic GUI, the APIC Advanced GUI, the APIC NX-OS style CLI, or through application calls to the APIC REST API.

Topology Supporting FCoE Initialization

In order for FCoE traffic flow to take place as described, you'll also need to set up separate VLAN connectivity over which SAN Hosts broadcast FCoE Initialization protocol (FIP) packets to discover the interfaces enabled as F ports.

vFC Interface Configuration Rules

Whether you set up the vFC network and EPG deployment through the APIC Basic or Advanced GUI, NX-OS style CLI, or the REST API, the following general rules apply across platforms:

- F port mode is the default mode for vFC ports. NP port mode must be specifically configured in the Interface policies.
- The load balancing default mode is for leaf-switch or interface level vFC configuration is src-dst-ox-id.
- One VSAN assignment per bridge domain is supported.
- The allocation mode for VSAN pools and VLAN pools must always be static.
- vFC ports require association with a VSAN domain (also called Fibre Channel domain) that contains VSANS mapped to VLANs.

Configuring FCoE Connectivity Using the REST API

You can configure FCoE-enabled interfaces and EPGs accessing those interfaces using the FCoE protocol with the REST API.

Procedure

-
- Step 1** To create a VSAN pool, send a post with XML such as the following example.
The example creates VSAN pool **vsanPool1** and specifies the range of VSANs to be included.

Example:

`https://apic-ip-address/api/mo/uni/infra/vsanns-[vsanPool1]-static.xml`

```
<!-- Vsan-pool -->
<fvnsVsanInstP name="vsanPool1" allocMode="static">
    <fvnsVsanEncapBlk name="encap" from="vsan-5" to="vsan-100"/>
</fvnsVsanInstP>
```

- Step 2** To create a VLAN pool, send a post with XML such as the following example.
The example creates VLAN pool **vlanPool1** and specifies the range of VLANs to be included.

Example:

```
https://apic-ip-address/api/mo/uni/infra/vlanns-[vlanPool1]-static.xml
```

```
<!-- Vlan-pool -->
<fvnsVlanInstP name="vlanPool1" allocMode="static">
  <fvnsEncapBlk name="encap" from="vlan-5" to="vlan-100"/>
</fvnsVlanInstP>
```

- Step 3** To create a VSAN-Attribute policy, send a post with XML such as the following example. The example creates VSAN attribute policy **vsanattr1**, maps **vsan-10** to **vlan-43**, and maps **vsan-11** to **vlan-44**.

Example:

```
https://apic-ip-address/api/mo/uni/infra/vsanattrp-[vsanattr1].xml
```

```
<fcVsanAttrP name="vsanattr1">

  <fcVsanAttrPEntry vlanEncap="vlan-43" vsanEncap="vsan-10"/>
  <fcVsanAttrPEntry vlanEncap="vlan-44" vsanEncap="vsan-11"
    lbType="src-dst-ox-id"/>
</fcVsanAttrP>
```

- Step 4** To create a Fibre Channel domain, send a post with XML such as the following example. The example creates VSAN domain **vsanDom1**.

Example:

```
https://apic-ip-address/api/mo/uni/fc-vsancDom1.xml
```

```
<!-- Vsan-domain -->
<fcDomP name="vsanDom1">
  <fcRsVsanAttr tDn="uni/infra/vsanattrp-[vsanattr1]"/>
  <infraRsVlanNs tDn="uni/infra/vlanns-[vlanPool1]-static"/>
  <fcRsVlanNs tDn="uni/infra/vsanns-[vsanPool1]-static"/>
</fcDomP>
```

- Step 5** To create the tenant, application profile, EPG and associate the FCoE bridge domain with the EPG, send a post with XML such as the following example. The example creates a bridge domain **bd1** under a target tenant configured to support FCoE and an application EPG **epg1**. It associates the EPG with VSAN domain **vsanDom1** and a Fibre Channel path (to interface **1/39** on leaf switch **101**). It deletes a Fibre channel path to interface **1/40** by assigning the **<fvRsFcPathAtt>** object with "deleted" status. Each interface is associated with a VSAN.

Note Two other possible alternative vFC deployments are also displayed. One sample deploys vFC on a port channel. The other sample deploys vFC on a virtual port channel.

Example:

```
https://apic-ip-address/api/mo/uni/tn-tenant1.xml
```

```
<fvTenant
  name="tenant1">
  <fvCtx name="vrf1"/>

  <!-- bridge domain -->
  <fvBD name="bd1" type="fc" >
    <fvRsCtx tnFvCtxName="vrf1" />
  </fvBD>

  <fvAp name="app1">
    <fvAEPg name="epg1">
      <fvRsBd tnFvBDName="bd1" />
      <fvRsDomAtt tDn="uni/fc-vsancDom1" />
      <fvRsFcPathAtt tDn="topology/pod-1/paths-101/pathep-[eth1/39]"
        vsan="vsan-11" vsanMode="native"/>
      <fvRsFcPathAtt tDn="topology/pod-1/paths-101/pathep-[eth1/40]" deleted="true"/>
    </fvAEPg>
  </fvAp>
</fvTenant>
```

```

    vsan="vsan-10" vsanMode="regular" status="deleted"/>
</fvAEPg>

<!-- Sample deployment of vFC on a port channel -->
<fvRsFcPathAtt vsanMode="native" vsan="vsan-10"
    tDn="topology/pod-1/paths 101/pathep-pc01"/>

<!-- Sample deployment of vFC on a virtual port channel -->
<fvRsFcPathAtt vsanMode="native" vsan="vsan-10"
    tDn="topology/pod-1/paths-101/pathep-vpc01"/>
<fvRsFcPathAtt vsanMode="native" vsan="vsan-10"
    tDn="topology/pod-1/paths-102/pathep-vpc01"/>

</fvAp>
</fvTenant>
```

- Step 6** To create a port policy group and an AEP, send a post with XML such as the following example. The example executes the following requests:

- Creates a policy group **portgrp1** that includes an FC interface policy **fcIfPol1**, a priority flow control policy **pfcIfPol1** and a slow-drain policy **sdIfPol1**.
- Creates an attached entity profile (AEP) **AttEntP1** that associates the ports in VSAN domain **vsanDom1** with the settings to be specified for **fcIfPol1**, **pfcIfPol1**, and **sdIfPol1**.

Example:

```

https://apic-ip-address/api/mo/uni.xml

<polUni>
    <infraInfra>
        <infraFuncP>
            <infraAccPortGrp name="portgrp1">
                <infraRsFcIfPol tnFcIfPolName="fcIfPol1"/>
                <infraRsAttEntP tDn="uni/infra/attentp-AttEntP1" />
                <infraRsQosPfcIfPol tnQosPfcIfPolName="pfcIfPol1"/>
                <infraRsQosSdIfPol tnQosSdIfPolName="sdIfPol1"/>
            </infraAccPortGrp>
        </infraFuncP>

        <infraAttEntityP name="AttEntP1">
            <infraRsDomP tDn="uni/fc-vsancDom1"/>
        </infraAttEntityP>
        <qosPfcIfPol dn="uni/infra/pfc-pfcIfPol1" adminSt="on">
        </qosPfcIfPol>
        <qosSdIfPol dn="uni/infra/qossdpol-sdIfPol1" congClearAction="log"
            congDetectMult="5" flushIntvl="100" flushAdminSt="enabled">
        </qosSdIfPol>
        <fcIfPol dn="uni/infra/fcIfPol-fcIfPol1" portMode="np">
        </fcIfPol>
    </infraInfra>
</polUni>
```

- Step 7** To create a node selector and a port selector, send a post with XML such as the following example. The example executes the following requests:

- Creates node selector **leafsel1** that specifies leaf node **101**.
- Creates port selector **portsel1** that specifies port **1/39**.

Example:
<https://apic-ip-address/api/mo/uni.xml>

```
<polUni>
  <infraInfra>
    <infraNodeP name="nprof1">
      <infraLeafS name="leafsel1" type="range">
        <infraNodeBlk name="nblk1" from_="101" to_="101"/>
      </infraLeafS>
      <infraRsAccPortP tDn="uni/infra/accportprof-pprof1"/>
    </infraNodeP>

    <infraAccPortP name="pprof1">
      <infraHPortS name="portsel1" type="range">
        <infraPortBlk name="blk"
          fromCard="1" toCard="1" fromPort="39" toPort="39">
        </infraPortBlk>

        <infraRsAccBaseGrp tDn="uni/infra/funcprof/accportgrp-portgrp1" />
      </infraHPortS>
    </infraAccPortP>
  </infraInfra>
</polUni>
```

Step 8 To create a vPC, send a post with XML such as the following example.

Example:
<https://apic-ip-address/api/mo/uni.xml>

```
<polUni>
  <fabricInst>
    <vpcInstPol name="vpc01" />

    <fabricProtPol pairT="explicit" >
      <fabricExplicitGEp name="vpc01" id="100" >
        <fabricNodePEp id="101"/>
        <fabricNodePEp id="102"/>
        <fabricRsVpcInstPol tnVpcInstPolName="vpc01" />
        <!-- <fabricLagId accBndlGrp="infraAccBndlGrp_{pcname}" /> -->
      </fabricExplicitGEp>
    </fabricProtPol>
  </fabricInst>
</polUni>
```

Configuring FCoE Over FEX Using REST API

Before You Begin

- Follow the steps 1 through 4 as described in [Configuring FCoE Connectivity Using the REST API, on page 194](#)

Procedure

Step 1 Configure FCoE over FEX (Selectors): Port:

Example:

```

<infraInfra dn="uni/infra">
  <infraNodeP name="nprof1">
    <infraLeafS name="leafsel1" type="range">
      <infraNodeBlk name="nblk1" from_="101" to_="101"/>
    </infraLeafS>
    <infraRsAccPortP tDn="uni/infra/accportprof-pprof1" />
  </infraNodeP>

  <infraAccPortP name="pprof1">
    <infraHPorts name="portsel1" type="range">
      <infraPortBlk name="blk"
        fromCard="1" toCard="1" fromPort="17" toPort="17"></infraPortBlk>
      <infraRsAccBaseGrp tDn="uni/infra/fexprof-fexprof1/fexbundle-fexbundle1" fexId="110"
    />
    </infraHPorts>
  </infraAccPortP>

  <infraFuncP>
    <infraAccPortGrp name="portgrp1">
      <infraRsAttEntP tDn="uni/infra/attentp-attentp1" />
    </infraAccPortGrp>
  </infraFuncP>

  <infraFexP name="fexprof1">
    <infraFexBndlGrp name="fexbundle1"/>
    <infraHPorts name="portsel2" type="range">
      <infraPortBlk name="blk2"
        fromCard="1" toCard="1" fromPort="20" toPort="20"></infraPortBlk>
      <infraRsAccBaseGrp tDn="uni/infra/funcprof/accportgrp-portgrp1"/>
    </infraHPorts>
  </infraFexP>

  <infraAttEntityP name="attentp1">
    <infraRsDomP tDn="uni/fc-vsancDom1"/>
  </infraAttEntityP>
</infraInfra>

```

Step 2 Tenant configuration:**Example:**

```

<fvTenant name="tenant1">
  <fvCtx name="vrf1"/>

  <!-- bridge domain -->
  <fvBD name="bd1" type="fc" >
    <fvRsCtx tnFvCtxName="vrf1" />
  </fvBD>

  <fvAp name="appl1">
    <fvAEPg name="epg1">
      <fvRsBd tnFvBDName="bd1" />
      <fvRsDomAtt tDn="uni/fc-vsancDom1" />
    <fvRsFcPathAtt tDn="topology/pod-1/paths-101/extpaths-110/pathep-[eth1/17]" vsan="vsan-11"
      vsanMode="native"/>
    </fvAEPg>
  </fvAp>
</fvTenant>

```

Step 3 Configure FCoE over FEX (Selectors): Port-Channel:**Example:**

```

<infraInfra dn="uni/infra">
  <infraNodeP name="nprof1">
    <infraLeafS name="leafsel1" type="range">
      <infraNodeBlk name="nblk1" from_="101" to_="101"/>
    </infraLeafS>
    <infraRsAccPortP tDn="uni/infra/accportprof-pprof1" />
  </infraNodeP>

```

```

</infraNodeP>

<infraAccPortP name="pprof1">
    <infraHPorts name="portsel1" type="range">
        <infraPortBlk name="blk1"
            fromCard="1" toCard="1" fromPort="18" toPort="18"></infraPortBlk>
        <infraRsAccBaseGrp tDn="uni/infra/fexprof-fexprof1/fexbundle1" fexId="111"
    />
    </infraHPorts>
</infraAccPortP>

<infraFexP name="fexprof1">
    <infraFexBndlGrp name="fexbundle1"/>
    <infraHPorts name="portsell" type="range">
        <infraPortBlk name="blk1"
            fromCard="1" toCard="1" fromPort="20" toPort="20"></infraPortBlk>
        <infraRsAccBaseGrp tDn="uni/infra/funcprof/accbundle-pc1"/>
    </infraHPorts>
</infraFexP>

<infraFuncP>
    <infraAccBndlGrp name="pc1">
        <infraRsAttEntP tDn="uni/infra/attentp-attentp1" />
    </infraAccBndlGrp>
</infraFuncP>

<infraAttEntityP name="attentp1">
<infraRsDomP tDn="uni/fc-vsancDom1"/>
</infraAttEntityP>
</infraInfra>

```

Step 4 Tenant configuration:

Example:

```

<fvTenant name="tenant1">
<fvCtx name="vrf1"/>

<!-- bridge domain -->
<fvBD name="bd1" type="fc" >
    <fvRsCtx tnFvCtxName="vrf1" />
</fvBD>

<fvAp name="app1">
    <fvAEPg name="epg1">
        <fvRsBd tnFvBDName="bd1" />
        <fvRsDomAtt tDn="uni/fc-vsancDom1" />
    <fvRsFcPathAtt tDn="topology/pod-1/paths-101/extpaths-111/pathp-[pc1]" vsan="vsan-11"
    vsanMode="native" />
        </fvAEPg>
    </fvAp>
</fvTenant>

```

Step 5 Configure FCoE over FEX (Selectors): VPC:

Example:

```

<polUni>
<fabricInst>
<vpcInstPol name="vpc1" />
<fabricProtPol pairT="explicit" >
    <fabricExplicitGEp name="vpc1" id="100" >
        <fabricNodePEp id="101"/>
        <fabricNodePEp id="102"/>
        <fabricRsVpcInstPol tnVpcInstPolName="vpc1" />
    </fabricExplicitGEp>
</fabricProtPol>
</fabricInst>
</polUni>

```

Step 6 Tenant configuration:

Example:

```
<fvTenant name="tenant1">
<fvCtx name="vrf1"/>

<!-- bridge domain -->
<fvBD name="bd1" type="fc" >
<fvRsCtx tnFvCtxName="vrf1" />
</fvBD>

<fvAp name="app1">
<fvAEPg name="epg1">
<fvRsBd tnFvBDName="bd1" />
<fvRsDomAtt tDn="uni/fc-vsancDom1" />
<fvRsFcPathAtt vsanMode="native" vsan="vsan-11"
tDn="topology/pod-1/protpaths-101-102/extprotpaths-111-111/pathp-[vpc1]" />
</fvAEPg>
</fvAp>
</fvTenant>
```

Step 7 Selector configuration:**Example:**

```
<polUni>
<infraInfra>
<infraNodeP name="nprof1">
<infraLeafS name="leafsel1" type="range">
<infraNodeBlk name="nblk1" from_="101" to_="101"/>
</infraLeafS>
<infraRsAccPortP tDn="uni/infra/accportprof-pprof1" />
</infraNodeP>

<infraNodeP name="nprof2">
<infraLeafS name="leafsel2" type="range">
<infraNodeBlk name="nblk2" from_="102" to_="102"/>
</infraLeafS>
<infraRsAccPortP tDn="uni/infra/accportprof-pprof2" />
</infraNodeP>

<infraAccPortP name="pprof1">
<infraHPortS name="portsel1" type="range">
<infraPortBlk name="blk1"
fromCard="1" toCard="1" fromPort="18" toPort="18">
</infraPortBlk>
<infraRsAccBaseGrp tDn="uni/infra/fexprof-fexprof1/fexbundle-fexbundle1" fexId="111" />
</infraHPortS>
</infraAccPortP>
<infraAccPortP name="pprof2">
<infraHPortS name="portsel2" type="range">
<infraPortBlk name="blk2"
fromCard="1" toCard="1" fromPort="18" toPort="18">
</infraPortBlk>
<infraRsAccBaseGrp tDn="uni/infra/fexprof-fexprof2/fexbundle-fexbundle2" fexId="111" />
</infraHPortS>
</infraAccPortP>

<infraFexP name="fexprof1">
<infraFexBndlGrp name="fexbundle1"/>
<infraHPortS name="portsel1" type="range">
<infraPortBlk name="blk1"
fromCard="1" toCard="1" fromPort="20" toPort="20">
</infraPortBlk>
<infraRsAccBaseGrp tDn="uni/infra/funcprof/accbundle-vpc1"/>
</infraHPortS>
</infraFexP>

<infraFexP name="fexprof2">
<infraFexBndlGrp name="fexbundle2"/>
<infraHPortS name="portsel2" type="range">
<infraPortBlk name="blk2"
```

```

fromCard="1" toCard="1" fromPort="20" toPort="20">
</infraPortBlk>
<infraRsAccBaseGrp tDn="uni/infra/funcprof/accbundle-vpc1"/>
</infraHPorts>
</infraFexP>

<infraFuncP>
<infraAccBndlGrp name="vpc1" lagT="node">
    <infraRsAttEntP tDn="uni/infra/attentp-attentp1" />
</infraAccBndlGrp>
</infraFuncP>

<infraAttEntityP name="attentp1">
<infraRsDomP tDn="uni/fc-vsандом1"/>
</infraAttEntityP>
</infraIntra>
</polUni>

```

Undeploying FCoE Connectivity through the REST API or SDK

To undeploy FCoE connectivity through the APIC REST API or SDK , delete the following objects associated with the deployment:

Object	Description
<fvRsFcPathAtt> (Fibre Channel Path)	The Fibre Channel path specifies the vFC path to the actual interface. Deleting each object of this type removes the deployment from that object's associated interfaces.
<fcVsanAttrpP> (VSAN/VLAN map)	The VSAN/VLAN map maps the VSANs to their associated VLANs deleting this object removes the association between the VSANs that support FCoE connectivity and their underlying VSANs.
<fvnsVsanInstP> (VSAN pool)	The VSAN pool specifies the set of VSANs available to support FCoE connectivity. Deleting this pool removes those VSANs.
<fvnsVlanInstP> ((VLAN pool)	The VLAN pool specifies the set of VLANs available for VSAN mapping. Deleting the associated VLAN pool cleans up after an FCoE undeployment, removing the underlying VLAN entities over which the VSAN entities ran.
<fcDomP> (VSAN or Fibre Channel domain)	The Fibre Channel domain includes all the VSANs and their mappings. Deleting this object undeploys vFC from all interfaces associated with this domain.
<fvAEPg> (application EPG)	The application EPG associated with the FCoE connectivity. If the purpose of the application EPGs was only to support FCoE-related activity, you might consider deleting this object.
<fvAp> (application profile)	The application profile associated with the FCoE connectivity. If the purpose of the application profile was only to support FCoE-related activity, you might consider deleting this object.

Object	Description
<fvTenant> (tenant)	The tenant associated with the FCoE connectivity. If the purpose of the tenant was only to support FCoE-related activity, you might consider deleting this object.

**Note**

If during clean up you delete the Ethernet configuration object (infraHPortS) for a vFC port, the default vFC properties remain associated with that interface. For example if the interface configuration for vFC NP port 1/20 is deleted, that port remains a vFC port but with default F port setting rather than non-default NP port setting applied.

The following steps undeploy FCoE-enabled interfaces and EPGs accessing those interfaces using the FCoE protocol.

Procedure

Step 1 To delete the associated Fibre Channel path objects, send a post with XML such as the following example. The example deletes all instances of the Fibre Channel path object <fvRsFcPathAtt>.

Note Deleting the Fibre Channel paths undeploys the vFC from the ports/VSANs that used them.

Example:

```
https://apic-ip-address/api/mo/uni/tn-tenant1.xml

<fvTenant
  name="tenant1">
  <fvCtx name="vrf1"/>

  <!-- bridge domain -->
  <fvBD name="bd1" type="fc" >
    <fvRsCtx tnFvCtxName="vrf1" />
  </fvBD>

  <fvAp name="app1">
    <fvAEPg name="epg1">
      <fvRsBd tnFvBDName="bd1" />
      <fvRsDomAtt tDn="uni/fc-vsандом1" />
      <fvRsFcPathAtt tDn="topology/pod-1/paths-101/pathеп-[eth1/39]"
        vsan="vsан-11" vsanMode="native" status="deleted"/>
      <fvRsFcPathAtt tDn="topology/pod-1/paths-101/pathеп-[eth1/40]"
        vsan="vsан-10" vsanMode="regular" status="deleted"/>
    </fvAEPg>

    <!-- Sample undeployment of vFC on a port channel -->
    <fvRsFcPathAtt vsanMode="native" vsan="vsан-10"
      tDн="topology/pod-1/paths 101/pathеп-pc01" status="deleted"/>

    <!-- Sample undeployment of vFC on a virtual port channel -->
    <fvRsFcPathAtt vsanMode="native" vsan="vsан-10"
      tDн="topology/pod-1/paths-101/pathеп-vpc01" status="deleted"/>
    <fvRsFcPathAtt vsanMode="native" vsan="vsан-10"
      tDн="topology/pod-1/paths-102/pathеп-vpc01" status="deleted"/>
```

```

</fvAp>
</fvTenant>
```

- Step 2** To delete the associated VSAN/VLAN map, send a post such as the following example. The example deletes the VSAN/VLAN map **vsanattr1** and its associated <fcVsanAttrP> object.

Example:

```

https://apic-ip-address/api/mo/uni/infra/vsanattrp-[vsanattr1].xml

<fcVsanAttrP name="vsanattr1" status="deleted"

```

- Step 3** To delete the associated VSAN pool, send a post such as the following example. The example deletes the VSAN pool **vsanPool1** and its associated <fvnsVsanInstP> object.

Example:

```

https://apic-ip-address/api/mo/uni/infra/vsanns-[vsanPool1]-static.xml

<!-- Vsan-pool -->
<fvnsVsanInstP name="vsanPool1" allocMode="static" status="deleted"

```

- Step 4** To delete the associated VLAN pool, send a post with XML such as the following example. The example deletes the VLAN pool **vlanPool1** and its associated <fvnsVlanInstP> object.

Example:

```

https://apic-ip-address/api/mo/uni/infra/vlanns-[vlanPool1]-static.xml

<!-- Vlan-pool -->
<fvnsVlanInstP name="vlanPool1" allocMode="static" status="deleted"

```

- Step 5** To delete the associated Fibre Channel domain, send a post with XML such as the following example. The example deletes the VSAN domain **vsanDom1** and its associated <fcDomP> object.

Example:

```

https://apic-ip-address/api/mo/uni/fc-vsандом1.xml
<!-- Vsan-domain -->
<fcDomP name="vsандом1" status="deleted"

```

- Step 6** **Optional:** If appropriate, you can delete the associated application EPG, the associated application profile, or the associated tenant.

Example:

In the following sample, the associated application EPG **epg1** and its associated <fvAEPg> object is deleted.

```

https://apic-ip-address/api/mo/uni/tn-tenant1.xml

<fvTenant
name="tenant1"/>
    <fvCtx name="vrf1"/>

    <!-- bridge domain -->
    <fvBD name="bd1" type="fc" >
```

```

<fvRsCtx tnFvCtxName="vrf1" />
</fvBD>

<fvAp name="app1">
  <fvAEPg name="epg1" status= "deleted">
    <fvRsBd tnFvBDName="bd1" />
    <fvRsDomAtt tDn="uni/fc-vsanDom1" />
    <fvRsFcPathAtt tDn="topology/pod-1/paths-101/pathep-[eth1/39]" vsan="vsan-11" vsanMode="native" status="deleted"/>
    <fvRsFcPathAtt tDn="topology/pod-1/paths-101/pathep-[eth1/40]" vsan="vsan-10" vsanMode="regular" status="deleted"/>
  </fvAEPg>

  <!-- Sample undeployment of vFC on a port channel -->
  <fvRsFcPathAtt vsanMode="native" vsan="vsan-10" tDN="topology/pod-1/paths 101/pathep-pc01" status="deleted"/>

  <!-- Sample undeployment of vFC on a virtual port channel -->
  <fvRsFcPathAtt vsanMode="native" vsan="vsan-10" tDn="topology/pod-1/paths-101/pathep-vpc01" status="deleted"/>
  <fvRsFcPathAtt vsanMode="native" vsan="vsan-10" tDn="topology/pod-1/paths-102/pathep-vpc01" status="deleted"/>

</fvAp>
</fvTenant>

```

Example:

In the following example, the associated application profile **app1** and its associated <fvAp> object is deleted.

<https://apic-ip-address/api/mo/uni/tn-tenant1.xml>

```

<fvTenant
  name="tenant1">
  <fvCtx name="vrf1"/>

  <!-- bridge domain -->
  <fvBD name="bd1" type="fc">
    <fvRsCtx tnFvCtxName="vrf1" />
  </fvBD>

  <fvAp name="app1" status="deleted">
    <fvAEPg name="epg1" status= "deleted">
      <fvRsBd tnFvBDName="bd1" />
      <fvRsDomAtt tDn="uni/fc-vsanDom1" />
      <fvRsFcPathAtt tDn="topology/pod-1/paths-101/pathep-[eth1/39]" vsan="vsan-11" vsanMode="native" status="deleted"/>
      <fvRsFcPathAtt tDn="topology/pod-1/paths-101/pathep-[eth1/40]" vsan="vsan-10" vsanMode="regular" status="deleted"/>
    </fvAEPg>

    <!-- Sample undeployment of vFC on a port channel -->
    <fvRsFcPathAtt vsanMode="native" vsan="vsan-10" tDN="topology/pod-1/paths 101/pathep-pc01" status="deleted"/>

    <!-- Sample undeployment of vFC on a virtual port channel -->
    <fvRsFcPathAtt vsanMode="native" vsan="vsan-10" tDn="topology/pod-1/paths-101/pathep-vpc01" status="deleted"/>
    <fvRsFcPathAtt vsanMode="native" vsan="vsan-10" tDn="topology/pod-1/paths-102/pathep-vpc01" status="deleted"/>

  </fvAp>
</fvTenant>

```

Example:

In the following example, the entire tenant **tenant1** and its associated <fvTenant> object is deleted.
<https://apic-ip-address/api/mo/uni/tn-tenant1.xml>

```
<fvTenant
  name="tenant1" status="deleted"

```

802.1Q Tunnels

About ACI 802.1Q Tunnels

With Cisco ACI and Cisco APIC Release 2.2(1x) and higher, you can configure 802.1Q tunnels to enable point-to-multi-point tunneling of Ethernet frames in the fabric, with Quality of Service (QoS) priority settings. A **dot1q-tunnel** transports untagged, 802.1Q tagged, and 802.1ad (QinQ) double-tagged frames as-is across the fabric. ACI front panel ports can be part of an **dot1q-tunnel** and function as QinQ edge-ports. Layer 2 switching is done based on Destination MAC (DMAC) and regular MAC learning is done in the tunnel.

The following guidelines and restrictions apply:

- **Dot1q-tunnels** are only supported on second-generation Cisco Nexus N9K switches (with "EX" on the end of the switch model name).
- Layer 2 tunneling of CDP, LACP, LLDP, and STP protocols is supported with the following restrictions:
 - Link Aggregation Control Protocol (LACP) tunneling functions as expected only with point-to-point tunnels using individual leaf interfaces. It is not supported on port-channels (PCs) or virtual port-channels (vPCs).
 - CDP and LLDP tunneling with PCs or vPCs is not deterministic; it depends on the link it chooses as the traffic destination.

- CDP and LLDP tunneling with more than two interfaces floods packets on all interfaces.
- ACI leaf switches do not perform any action in response to STP TCN packets and TCN packets can only be tunneled.
- If a PC or VPC is the only interface in a **dot1q-tunnel** and it is **deleted** and reconfigured, remove the association of the PC/VPC to the **dot1q-tunnel** and reconfigure it.
- The Ethertypes for double-tagged frames must be 0x9100 followed by 0x8100.
- You can configure multiple ports (even across leaf switches) in a **dot1q-tunnel**. A port may only be part of one tunnel.
- Regular EPGs are not supported in tunnels.
- FEX interfaces are not supported as members of a **dot1q-tunnel**.
- Interface-level statistics are supported for interfaces in **dot1q-tunnels**, but statistics at the tunnel level are not supported.

Configuring 802.1Q Tunnels With Ports Using the REST API

Create a **dot1q-tunnel**, using ports, and configure the interfaces for it with steps such as the following examples.

Before You Begin

Configure the tenant that will use the **dot1q-tunnel**.

Procedure

Step 1 Create a **dot1q-tunnel** using the REST API with XML such as the following:

Example:

```
<fvTn1EPg name="VRF64_dot1q_tunnel" qiqL2ProtTunMask="lldp" >
  <fvRsTnlpathAtt tDn="topology/pod-1/paths-101/pathep-[eth1/13]" />
</fvTn1EPg>
```

Step 2 Configure a Layer 2 Interface policy with static binding with XML such as the following:

Example:

```
<l2IfPol name="VRF64_L2_int_pol" qinq="edgePort" />
```

Step 3 Apply the Layer 2 Interface policy to a Leaf Access Port Policy Group with XML such as the following:

Example:

```
<infraAccPortGrp name="VRF64_L2_Port_Pol_Group" >
  <infraRsL2IfPol tnL2IfPolName="VRF64_L2_int_pol"/>
</infraAccPortGrp>
```

Step 4 Configure a Leaf Profile with an Interface Selector with XML such as the following example:

Example:

```
<infraAccPortP name="VRF64_dot1q_leaf_profile" >
  <infraHPorts name="vrf64_access_port_selector" type="range">
    <infraPortBlk name="block2" toPort="15" toCard="1" fromPort="13" fromCard="1"/>
    <infraRsAccBaseGrp tDn="uni/infra/funcprof/accportgrp-VRF64_L2_Port_Pol_Group" />
```

```

        </infraHPortS>
</infraAccPortP>

```

The following example shows the port configuration in two posts:

XML with Post 1:

```

<polUni>
    <infraInfra>
        <l2IfPol name="testL2IfPol" qinq="1"/>
        <infraNodeP name="Node_101_phys">
            <infraLeafS name="phys101" type="range">
                <infraNodeBlk name="test" from_="101" to_="101"/>
            </infraLeafS>
            <infraRsAccPortP tDn="uni/infra/accportprof-phys21"/>
        </infraNodeP>
        <infraAccPortP name="phys21">
            <infraHPortS name="physHPorts" type="range">
                <infraPortBlk name="phys21" fromCard="1" toCard="1" fromPort="21" toPort="21"/>
                <infraRsAccBaseGrp tDn="uni/infra/funcprof/accportgrp-21"/>
            </infraHPortS>
        </infraAccPortP>
        <infraFuncP>
            <infraAccPortGrp name="21">
                <infraRsL2IfPol tnL2IfPolName="testL2IfPol"/>
                <infraRsAttEntP tDn="uni/infra/attentp-AttEntityProf1701"/>
            </infraAccPortGrp>
        </infraFuncP>
        <l2IfPol name='testL2IfPol' qinq='edgePort' />
        <infraAttEntityP name="AttEntityProf1701">
            <infraRsDomP tDn="uni/phys-dom1701"/>
        </infraAttEntityP>
    </infraInfra>
</polUni>

```

XML with Post 2:

```

<polUni>
    <fvTenant dn="uni/tn-Coke" name="Coke">
        <fvTn1EPg name="WEB5">
            <fvRsTnlpathAtt tDn="topology/pod-1/paths-101/pathep-[eth1/21]"/>
        </fvTn1EPg>
    </fvTenant>
</polUni>

```

Configuring 802.1Q Tunnels With PCs Using the REST API

Create a **dot1q-tunnel**, using PCs, and configure the interfaces for it with steps such as the following examples.

Before You Begin

Configure the tenant that will use the **dot1q-tunnel**.

Procedure

Step 1 Create an **dot1q-tunnel** using the REST API with XML such as the following:

Example:

```

<fvTn1EPg name="WEB" qiqL2ProtTunMask=lldp>
    <fvRsTnlpathAtt tDn="topology/pod-1/paths-101/pathep-[po2]"/>
</fvTn1EPg>

```

Step 2 Configure a Layer 2 Interface policy with static binding with XML such as the following:

Example:

```
<l2IfPol name="testL2IfPol" qinq="edgePort"/>
```

Step 3 Apply the Layer 2 Interface policy to a Leaf Access Port Policy Group with XML such as the following:

Example:

```
<infraAccBndlGrp name="po2" lagT="link">
    <infraRsL2IfPol tnL2IfPolName="testL2IfPol"/>
</infraAccBndlGrp>
```

Step 4 Configure a Leaf Profile with an Interface Selector with XML such as the following example:

Example:

```
<infraAccPortP name="PC">
    <infraHPortS name="allow" type="range">
        <infraPortBlk name="block2" fromCard="1" toCard="1" fromPort="10" toPort="11" />
        <infraRsAccBaseGrp tDn="uni/infra/funcprof/accbundle-po2"/>
    </infraHPortS>
</infraAccPortP>
```

The following example shows the port-channel configuration in two posts:

XML with Post 1:

```
<infraInfra dn="uni/infra">
    <infraNodeP name="bLeaf3">
        <infraLeafS name="leafs3" type="range">
            <infraNodeBlk name="nblk3" from_="101" to_="101">
            </infraNodeBlk>
        </infraLeafS>
        <infraRsAccPortP tDn="uni/infra/accportprof-shipping3"/>
    </infraNodeP>
    <infraAccPortP name="shipping3">
        <infraHPorts name="pselc3" type="range">
            <infraPortBlk name="blk3" fromCard="1" toCard="1" fromPort="24" toPort="25"/>

            <infraRsAccBaseGrp tDn="uni/infra/funcprof/accbundle-accountingLag3" />
        </infraHPortS>
    </infraAccPortP>
    <infraFuncP>
        <infraAccBndlGrp name="accountingLag3" lagT='link'>
            <infraRsAttEntP tDn="uni/infra/attentp-default"/>
                <infraRsLacpPol tnLacpLagPolName='accountingLacp3' />
                <infraRsL2IfPol tnL2IfPolName="testL2IfPol3"/>
        </infraAccBndlGrp>
    </infraFuncP>
    <lacpLagPol name='accountingLacp3' ctrl='15' descr='accounting' maxLinks='14' minLinks='1' mode='active' />
    <l2IfPol name='testL2IfPol3' qinq='edgePort' />
    <infraAttEntityP name="default">
    </infraAttEntityP>
</infraInfra>
```

XML with Post 2:

```
<polUni>
    <fvTenant dn="uni/tn-Coke" name="Coke">
        <!-- bridge domain -->
        <fvTn1EPg name="WEB6">
            <fvRsTnlpathAtt tDn="topology/pod-1/paths-101/pathep-[accountingLag1]" />
        </fvTn1EPg>
    </fvTenant>
</polUni>
```

Configuring 802.1 Q Tunnels With VPCs Using the REST API

Create a **dot1q-tunnel**, using VPCs, and configure the interfaces for it with steps such as the following examples.

Before You Begin

Configure the tenant that will use the **dot1q-tunnel**.

Procedure

- Step 1** Create an 802.1Q tunnel using the REST API with XML such as the following:

Example:

```
<fvTnlEPg name="WEB" qiqL2ProtTunMask=lldp>
    <fvRsTnlpathAtt tDn="topology/pod-1/protpaths-101-102/pathep-[po4]" />
</fvTnlEPg>
```

- Step 2** Configure a Layer 2 Interface policy with static binding with XML such as the following:

Example:

```
<l2IfPol name="testL2IfPol" qinq="edgePort"/>
```

- Step 3** Apply the Layer 2 Interface policy to a Leaf Access Port Policy Group with XML such as the following:

Example:

```
<infraAccBndlGrp name="po4" lagT="node">
    <infraRsL2IfPol tnL2IfPolName="testL2IfPol"/>
</infraAccBndlGrp>
```

- Step 4** Configure a Leaf Profile with an Interface Selector with XML such as the following example:

Example:

```
<infraAccPortP name="VPC">
    <infraHPorts name="allow" type="range">
        <infraPortBlk name="block2" fromCard="1" toCard="1" fromPort="10" toPort="11" />
        <infraRsAccBaseGrp tDn="uni/infra/funcprof/accbundle-po4"/>
    </infraHPorts>
</infraAccPortP>
```

The following example shows the VPC configuration in three posts:

XML with Post 1:

```
<polUni>
    <fabricInst>
        <fabricProtPol pairT="explicit">
            <fabricExplicitGEp name="101-102-vpc1" id="30">
                <fabricNodePEP id="101"/>
                <fabricNodePEP id="102"/>
            </fabricExplicitGEp>
        </fabricProtPol>
    </fabricInst>
</polUni>
```

XML with Post 2:

```
<infraInfra dn="uni/infra">
    <infraNodeP name="bLeaf1">
        <infraLeafS name="leafs" type="range">
            <infraNodeBlk name="nblk" from_="101" to_="101">
        </infraNodeBlk>
```

```

        </infraLeafS>
        <infraRsAccPortP tDn="uni/infra/accportprof-shipping1"/>
    </infraNodeP>

    <infraNodeP name="bLeaf2">
        <infraLeafS name="leafs" type="range">
            <infraNodeBlk name="nblk" from_="102" to_="102">
            </infraNodeBlk>
        </infraLeafS>
        <infraRsAccPortP tDn="uni/infra/accportprof-shipping2"/>
    </infraNodeP>

    <infraAccPortP name="shipping1">
        <infraHPortS name="pselc" type="range">
            <infraPortBlk name="blk" fromCard="1" toCard="1" fromPort="4" toPort="4"/>
            <infraRsAccBaseGrp tDn="uni/infra/funcprof/accbundle-accountingLag1" />
        </infraHPortS>
    </infraAccPortP>

    <infraAccPortP name="shipping2">
        <infraHPortS name="pselc" type="range">
            <infraPortBlk name="blk" fromCard="1" toCard="1" fromPort="2" toPort="2"/>
            <infraRsAccBaseGrp tDn="uni/infra/funcprof/accbundle-accountingLag2" />
        </infraHPortS>
    </infraAccPortP>

    <infraFuncP>
        <infraAccBndlGrp name="accountingLag1" lagT='node'>
            <infraRsAttEntP tDn="uni/infra/attentp-default"/>
            <infraRsLacpPol tnLacpLagPolName='accountingLacp1' />
            <infraRsL2IfPol tnL2IfPolName="testL2IfPol"/>
        </infraAccBndlGrp>
        <infraAccBndlGrp name="accountingLag2" lagT='node'>
            <infraRsAttEntP tDn="uni/infra/attentp-default"/>
            <infraRsLacpPol tnLacpLagPolName='accountingLacp1' />
            <infraRsL2IfPol tnL2IfPolName="testL2IfPol"/>
        </infraAccBndlGrp>
    </infraFuncP>
    <lacpLagPol name='accountingLacp1' ctrl='15' descr='accounting' maxLinks='14' minLinks='1' mode='active' />
    <l2IfPol name='testL2IfPol' qinq='edgePort' />

    <infraAttEntityP name="default">
    </infraAttEntityP>
</infraInfra>

```

XML with Post 3:

```

<polUni>
    <fvTenant dn="uni/tn-Coke" name="Coke">
        <!-- bridge domain -->
        <fvTn1EPg name="WEB6">
            <fvRsTn1pathAtt tDn="topology/pod-1/protpaths-101-102/pathep-[accountingLag2]" />
        </fvTn1EPg>
    </fvTenant>
</polUni>

```

Breakout Ports

Configuration of Dynamic Breakout Ports

To enable a 40 Gigabit Ethernet (GE) leaf switch port to be connected to 4-10GE capable (downlink) devices (connected with Cisco 40-Gigabit to 4X10-Gigabit breakout cables), you configure the 40GE port to breakout (split) to 4-10GE ports.

**Note**

This feature is supported only on the access facing ports of the N9K-C9332PQ switch.
100GE breakout ports are currently not supported.

Observe the following guidelines and restrictions:

- You can configure ports 1 to 26 as downlink ports. Of those ports, breakout ports can be configured on port 1 to 12 and 15 to 26. Ports 13 and 14 do not support breakout.
- Breakout subports can be used in the same way other port types in the policy model are used.
- When a port is enabled for dynamic breakout, other policies (expect monitoring policies) on the parent port are no longer valid.
- When a port is enabled for dynamic breakout, other EPG deployments on the parent port are no longer valid.
- A breakout sub-port can not be further broken out using a breakout policy group.

You can configure 40GE ports for dynamic breakout using the Basic or Advanced mode APIC GUI, the NX-OS style CLI, or the REST API.

Configuring Dynamic Breakout Ports Using the REST API

Configure a Breakout Leaf Port with an Leaf Interface Profile, associate the profile with a switch, and configure the sub ports with the following steps.

Procedure

Before You Begin

- The ACI fabric is installed, APIC controllers are online, and the APIC cluster is formed and healthy.
- An APIC fabric administrator account is available that will enable creating the necessary fabric infrastructure configurations.
- The target leaf switches are registered in the ACI fabric and available.
- The 40GE leaf switch ports are connected with Cisco breakout cables to the downlink ports.

Procedure

Step 1 Configure a breakout policy group for the breakout port with JSON, such as the following example:

Example:

In this example, we create an interface profile 'brkouttest' with the only port 44 underneath its port selector. The port selector is pointed to a breakout policy group 'new-brkoutPol'.

```
{
    "infraAccPortP": {
        "attributes": {
            "dn": "uni/infra/accportprof-brkouttest",
            "name": "brkouttest",
            "rn": "accportprof-brkouttest",
            "status": "created,modified"
        }
    }
}
```

```
        },
        "children": [
            {
                "infraHPortsS": {
                    "attributes": {
                        "dn": "uni/infra/accportprof-brkouttest/hports-tst1-typ-range",
                        "name": "tst1",
                        "rn": "hports-tst1-typ-range",
                        "status": "created,modified"
                    },
                    "children": [
                        {
                            "infraPortBlk": {
                                "attributes": {

```

"dn": "uni/infra/accportprof-brkouttest/hports-tst1-typ-range/portblk-block2",
"fromPort": "44",
"toPort": "44",
"name": "block2",
"rn": "portblk-block2",
"status": "created,modified"
},
"children": [] }
}, {
"infraRsAccBaseGrp": {
"attributes": {
"tDn": "uni/infra/funcprof/brkoutportgrp-new-brkoutPol",
"status": "created,modified"
},
"children": []
}
}
]
}
]
}

Step 2 Create a new switch profile and associate it with the port profile, previously created, with JSON such as the following example:

Example:

In this example, we create a new switch profile 'leaf1017' with switch 1017 as the only node. We associate this new switch profile with the port profile 'brkouttest' created above. After this, the port 44 on switch 1017 will have 4 sub ports.

Example:

```
{
  "infraNodeP": {
    "attributes": {
      "dn": "uni/infra/nprof-leaf1017",
      "name": "leaf1017",
      "rn": "nprof-leaf1017",
      "status": "created,modified"
    },
    "children": [ {
      "infraLeafs": {
        "attributes": {
          "dn": "uni/infra/nprof-leaf1017/leaves-1017-typ-range",
          "type": "range",
          "name": "1017",
          "rn": "leaves-1017-typ-range",
          "status": "created"
        },
        "children": [ {
          "infraNodeBlk": {
            "attributes": {
              "dn": "uni/infra/nprof-leaf1017/leaves-1017-typ-range/nodeblk-102bf7dc60e63f7e",
              "from_": "1017",
              "to_": "1017"
            }
          }
        } ]
      }
    } ]
  }
}
```

```

        "name":"102bf7dc60e63f7e",
        "rn":"nodeblk-102bf7dc60e63f7e",
        "status":"created"
    },
    "children": []
}
]
},
{
    "infraRsAccPortP": {
        "attributes": {
            "tDn":"uni/infra/accportprof-brkouttest",
            "status":"created,modified"
        },
        "children": []
    }
]
}
]
```

Step 3 Configure the subports.

Example:

This example configures subports 1/44/1, 1/44/2, 1/44/3, 1/44/4 on switch 1017, for instance, in the example below, we configure interface 1/41/3. It also creates the `infraSubPortBlk` object instead of the `infraPortBlk` object.

```
{
    "infraAccPortP": {
        "attributes": {
            "dn":"uni/infra/accportprof-brkouttest1",
            "name":"brkouttest1",
            "rn":"accportprof-brkouttest1",
            "status":"created,modified"
        },
        "children": [
            "infraHPortS": {
                "attributes": {
                    "dn":"uni/infra/accportprof-brkouttest1/hports-sell-typ-range",
                    "name":"sell1",
                    "rn":"hports-sell-typ-range",
                    "status":"created,modified"
                },
                "children": [
                    {
                        "infraSubPortBlk": {
                            "attributes": {
                                "dn":"uni/infra/accportprof-brkouttest1/hports-sell-typ-range/subportblk-block2",
                                "fromPort":"44",
                                "toPort":"44",
                                "fromSubPort":"3",
                                "toSubPort":"3",
                                "name":"block2",
                                "rn":"subportblk-block2",
                                "status":"created"
                            },
                            "children": []
                        }
                    },
                    {
                        "infraRsAccBaseGrp": {
                            "attributes": {
                                "tDn":"uni/infra/funcprof/accportgrp-p1",
                                "status":"created,modified"
                            },
                            "children": []
                        }
                    }
                ]
            }
        ]
    }
}
```

```
}
```

IGMP Snooping

About Cisco APIC and IGMP Snooping

IGMP snooping is the process of listening to Internet Group Management Protocol (IGMP) network traffic. The feature allows a network switch to listen in on the IGMP conversation between hosts and routers and filter multicasts links that do not need them, thus controlling which ports receive specific multicast traffic.

Cisco APIC provides support for the full IGMP snooping feature included on a traditional switch such as the N9000 standalone.

- Policy-based IGMP snooping configuration per bridge domain

APIC enables you to configure a policy in which you enable, disable, or customize the properties of IGMP Snooping on a per bridge-domain basis. You can then apply that policy to one or multiple bridge domains.

- Static port group implementation

IGMP static port grouping enables you to pre-provision ports, already statically-assigned to an application EPG, as the switch ports to receive and process IGMP multicast traffic. This pre-provisioning prevents the join latency which normally occurs when the IGMP snooping stack learns ports dynamically.

Static group membership can be pre-provisioned only on static ports (also called, *static-binding ports*) assigned to an application EPG.

- Access group configuration for application EPGs

An “access-group” is used to control what streams can be joined behind a given port.

An access-group configuration can be applied on interfaces that are statically assigned to an application EPG in order to ensure that the configuration can be applied on ports that will actually belong to the that EPG.

Only Route-map-based access groups are allowed.

How IGMP Snooping is Implemented in the ACI Fabric



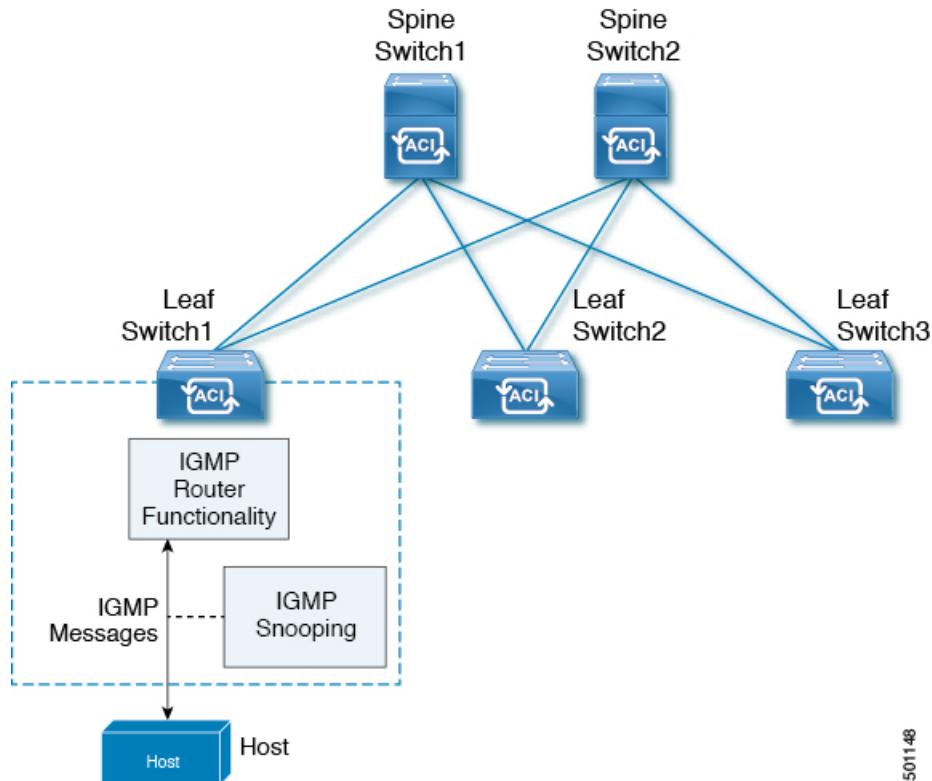
Note

We recommend that you do not disable IGMP snooping on the bridge domain. If you disable IGMP snooping, you might see reduced multicast performance because of excessive false flooding within the bridge domain.

IGMP snooping software examines Layer 2 IP multicast traffic within a bridge domain to discover the ports where interested receivers reside. Using the port information, IGMP snooping can reduce bandwidth consumption in a multi-access bridge domain environment to avoid flooding the entire bridge domain. By default, IGMP snooping is enabled on the bridge domain.

This figure shows the IGMP routing functions and IGMP snooping functions both contained on an ACI leaf switch with connectivity to a host. The IGMP snooping feature snoops the IGMP membership reports and Leave messages and forwards them only when necessary to the IGMP router function.

Figure 21: IGMP Snooping function



501148

IGMP snooping operates upon IGMPv1, IGMPv2, and IGMPv3 control plane packets where Layer 3 control plane packets are intercepted and influence the Layer 2 forwarding behavior.

IGMP snooping has the following proprietary features:

- Source filtering that allows forwarding of multicast packets based on destination and source IP addresses
- Multicast forwarding based on IP addresses rather than the MAC address
- Multicast forwarding alternately based on the MAC address



Note

For more information about IGMP snooping, see RFC 4541.

Virtualization Support

You can define multiple virtual routing and forwarding (VRF) instances for IGMP snooping.

On leaf switches, you can use the **show** commands with a VRF argument to provide a context for the information displayed. The default VRF is used if no VRF argument is supplied.

Configuring and Assigning an IGMP Snoop Policy to a Bridge Domain using the REST API

Procedure

To configure an IGMP Snooping policy and assign it to a bridge domain, send a post with XML such as the following example:

Example:

```
https://apic-ip-address/api/node/mo/uni/.xml
<fvTenant name="mcast_tenant1">

    <!-- Create an IGMP snooping template, and provide the options -->
    <igmpSnoopPol name="igmp_snp_bd_21"
        adminSt="enabled"
        lastMbrIntvl="1"
        queryIntvl="125"
        rspIntvl="10"
        startQueryCnt="2"
        startQueryIntvl="31"
    />
    <fvCtx name="ip_video"/>

    <fvBD name="bd_21">
        <fvRsCtx tnFvCtxName="ip_video"/>

        <!-- Bind igmp snooping to a BD -->
        <fvRsIgmpsn tnIgmpSnoopPolName="igmp_snp_bd_21"/>
    </fvBD></fvTenant>
```

This example creates and configures the the IGMP Snoop policy, igmp_snp_bd_12 with the following properties, and binds the IGMP policy, igmp_snp_bd_21, to bridge domain, bd_21:

- Administrative state is enabled
- Last Member Query Interval is the default 1 second.
- Query Interval is the default 125.
- Query Response interval is the default 10 seconds
- The Start Query Count is the default 2 messages
- The Start Query interval is 35 seconds.

Enabling Group Access to IGMP Layer 2 Multicast using REST API

After you have enabled IGMP snooping and Layer 2 multicasting on ports that have been statically assigned to an EPG, you can then create and assign access groups of users that are permitted or denied access to the IGMP snooping and multicast traffic enabled on those ports.

Procedure

Define the access group, "foobroker."

The following example sequence configures: Access group "foobroker" associated with tenant_A, Rmap_A, application_A, epg_A, on leaf 102, interface 1/10, VLAN 202. By association with Rmap_A, the access group "foobroker" has access to multicast traffic received at multicast address 226.1.1.1/24 and is denied access to traffic received at multicast address 227.1.1.1/24.

Example:

```
<!-- api/node/mo/uni/.xml -->
<fvTenant name="tenant_A">
  <pimRouteMapPol name="Rmap_A">
    <pimRouteMapEntry action="permit" grp="226.1.1.1/24" order="10"/>
    <pimRouteMapEntry action="deny" grp="227.1.1.1/24" order="20"/>
  </pimRouteMapPol>
  <fvAp name="application_A">
    <fvAEPg name="epg_A">
      <fvRsPathAtt encap="vlan-202" instrImedcy="immediate" mode="regular"
tDn="topology/pod-1/paths-102/pathep-[eth1/10]">
        <!-- IGMP snooping access group case -->
        <igmpSnoopAccessGroup name="foobroker">
          <igmpRsSnoopAccessGroupFilterRMap tnPimRouteMapPolName="Rmap_A"/>
        </igmpSnoopAccessGroup>
      </fvRsPathAtt>
    </fvAEPg>
  </fvAp>
</fvTenant>
```

Enabling IGMP Layer 2 Multicast on Static Ports Using the REST API

You can enable IGMP snoop and layer 2 multicast processing on ports that have been statically assigned to an EPG. You can create and assign access groups of users that are permitted or denied access to the IGMP snoop and multicast traffic enabled on those ports.

Before You Begin**Procedure**

To configure application EPGs with static ports, enable those ports to receive and process IGMP snoop and layer 2 multicast traffic, and assign groups to access or be denied access to that traffic, send a post with XML such as the following example.

In the following example, IGMP snoop is enabled on leaf 102 interface 1/10 on VLAN 202. Multicast IP addresses 224.1.1.1 and 225.1.1.1 are associated with this port.

Example:

```
https://apic-ip-address/api/node/mo/uni/.xml
<fvTenant name="tenant_A">
  <fvAp name="appplcation_A">
    <fvAEPg name="epg_A">
      <fvRsPathAtt encap="vlan-202" instrImedcy="immediate" mode="regular"
tDn="topology/pod-1/paths-102/pathep-[eth1/10]">
        <!-- IGMP snooping static group case -->
        <igmpSnoopStaticGroup group="224.1.1.1" source="0.0.0.0"/>
        <igmpSnoopStaticGroup group="225.1.1.1" source="2.2.2.2"/>
      </fvRsPathAtt>
    </fvAEPg>
  </fvAp>
</fvTenant>
```

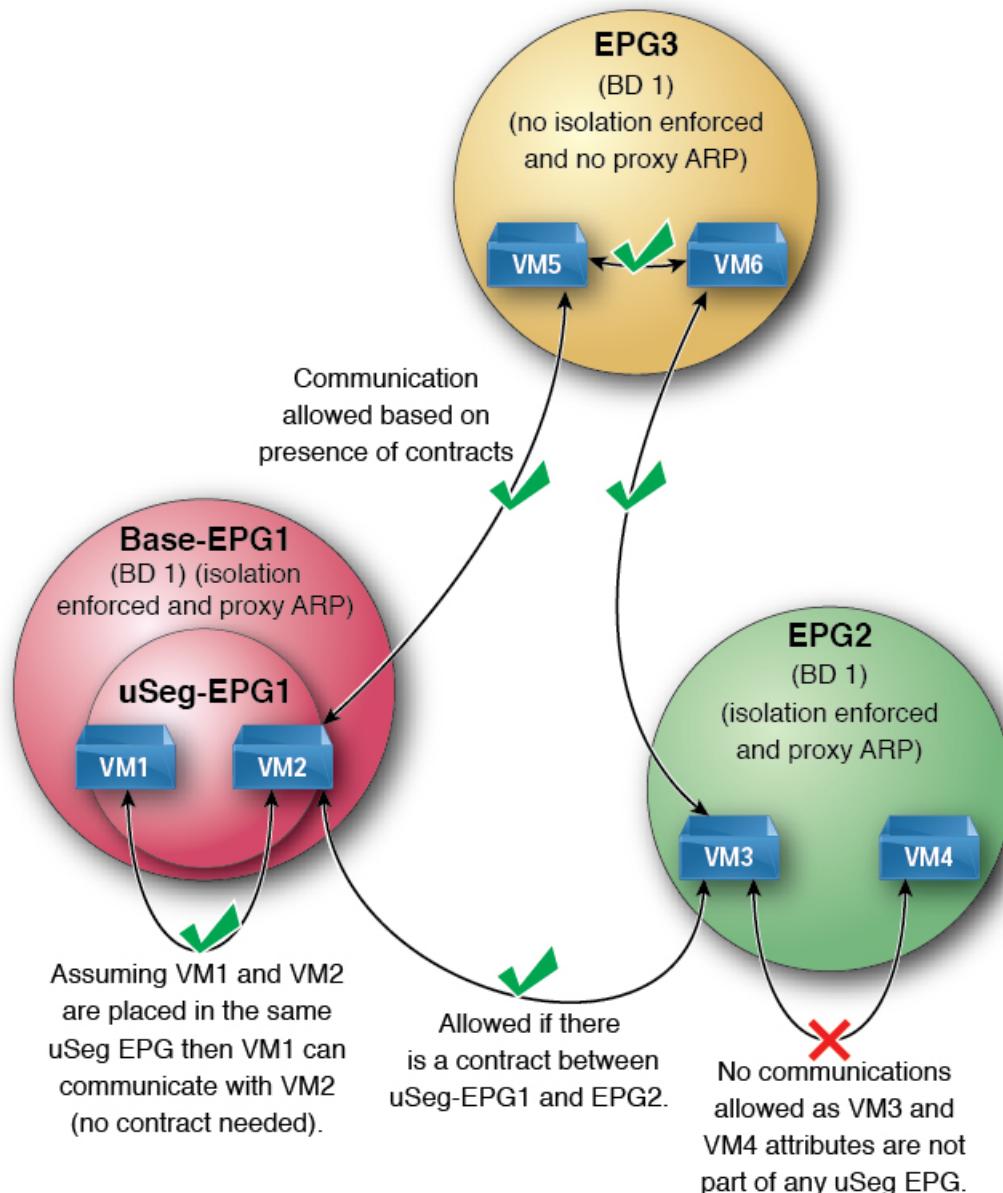
Proxy ARP

About Proxy ARP

Proxy ARP in Cisco ACI enables endpoints within a network or subnet to communicate with other endpoints without knowing the real MAC address of the endpoints. Proxy ARP is aware of the location of the traffic destination, and offers its own MAC address as the final destination instead.

To enable Proxy ARP, intra-EPG endpoint isolation must be enabled on the EPG see the following figure for details. For more information about intra-EPG isolation and Cisco ACI, see the *Cisco ACI Virtualization Guide*.

Figure 22: Proxy ARP and Cisco APIC



Proxy ARP within the Cisco ACI fabric is different from the traditional proxy ARP. As an example of the communication process, when proxy ARP is enabled on an EPG, if an endpoint A sends an ARP request for endpoint B and if endpoint B is learned within the fabric, then endpoint A will receive a proxy ARP response from the bridge domain (BD) MAC. If endpoint A sends an ARP request for endpoint B, and if endpoint B is not learned within the ACI fabric already, then the fabric will send a proxy ARP request within the BD. Endpoint B will respond to this proxy ARP request back to the fabric. At this point, the fabric does not send

a proxy ARP response to endpoint A, but endpoint B is learned within the fabric. If endpoint A sends another ARP request to endpoint B, then the fabric will send a proxy ARP response from the BD MAC.

The following example describes the proxy ARP resolution steps for communication between clients VM1 and VM2:

- 1 VM1 to VM2 communication is desired.

Figure 23: VM1 to VM2 Communication is Desired.

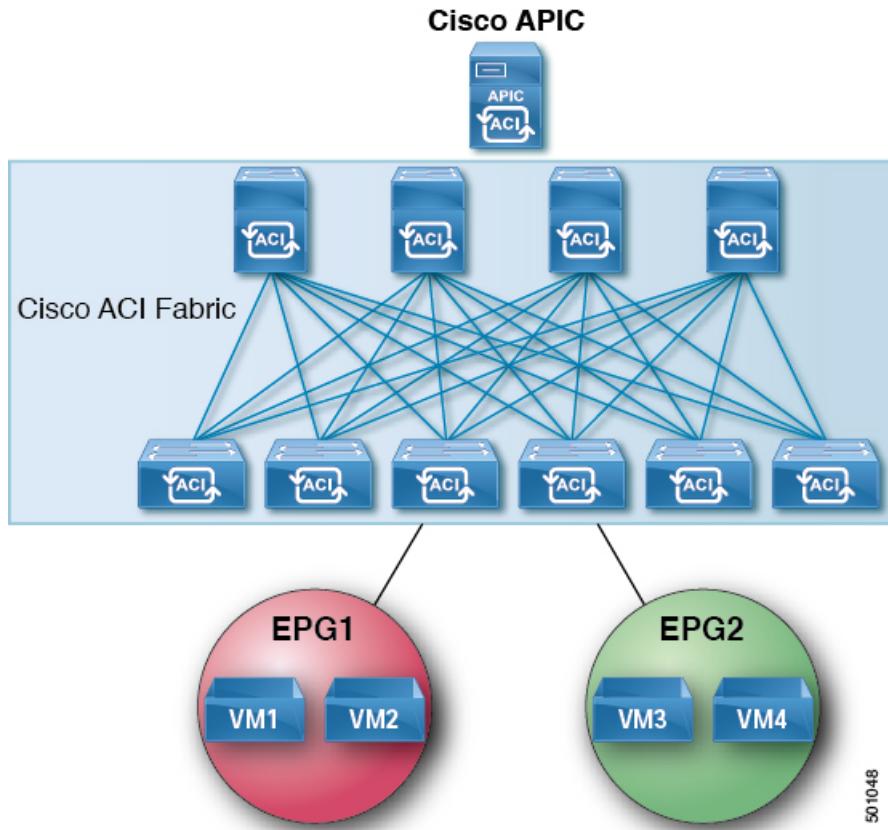


Table 4: ARP Table State

Device	State
VM1	IP = * MAC = *
ACI fabric	IP = * MAC = *
VM2	IP = * MAC = *

- 2 VM1 sends an ARP request with a broadcast MAC address to VM2.

Figure 24: VM1 sends an ARP Request with a Broadcast MAC address to VM2

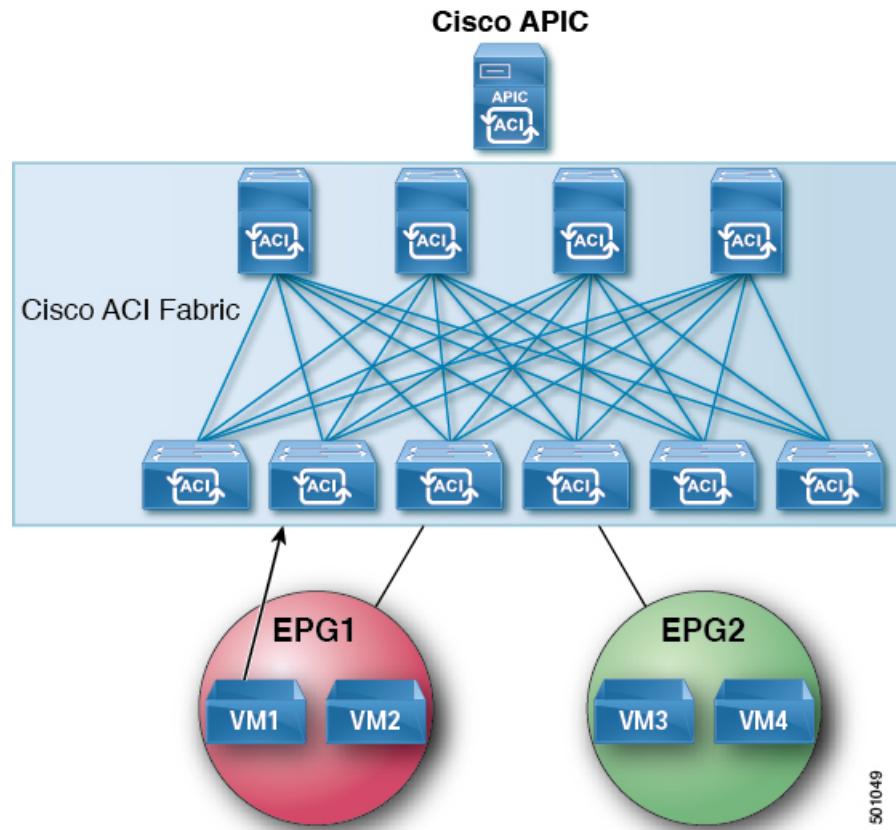


Table 5: ARP Table State

Device	State
VM1	IP = VM2 IP; MAC = ?
ACI fabric	IP = VM1 IP; MAC = VM1 MAC
VM2	IP = * MAC = *

- 3 The ACI fabric floods the proxy ARP request within the bridge domain (BD).

Figure 25: ACI Fabric Floods the Proxy ARP Request within the BD

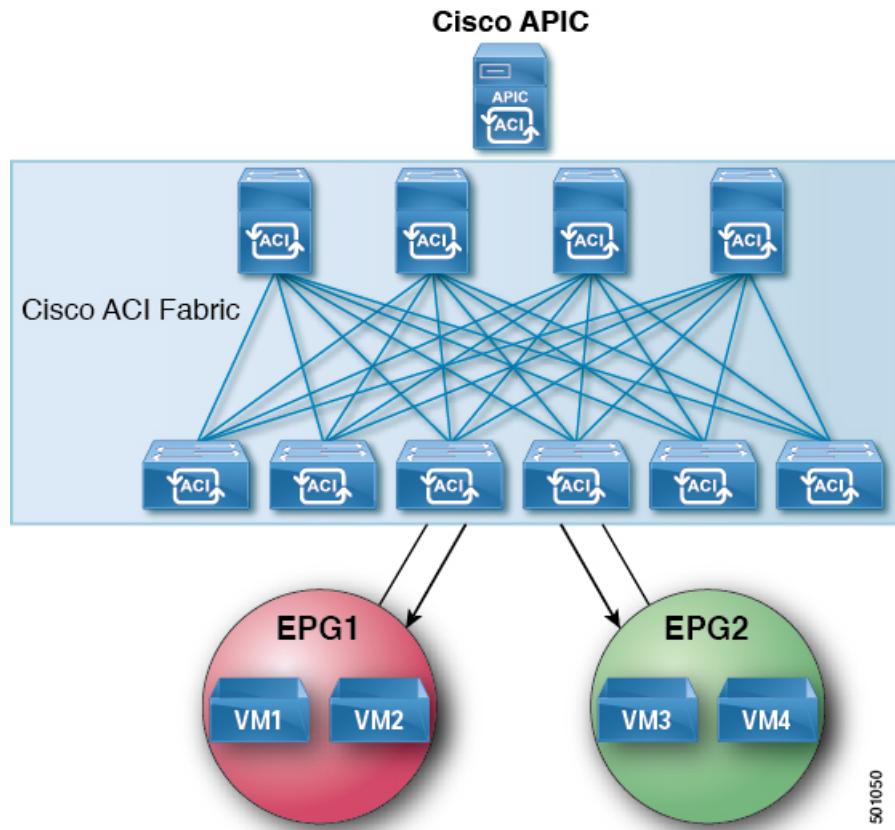


Table 6: ARP Table State

Device	State
VM1	IP = VM2 IP; MAC = ?
ACI fabric	IP = VM1 IP; MAC = VM1 MAC
VM2	IP = VM1 IP; MAC = BD MAC

- 4 VM2 sends an ARP response to the ACI fabric.

Figure 26: VM2 Sends an ARP Response to the ACI Fabric

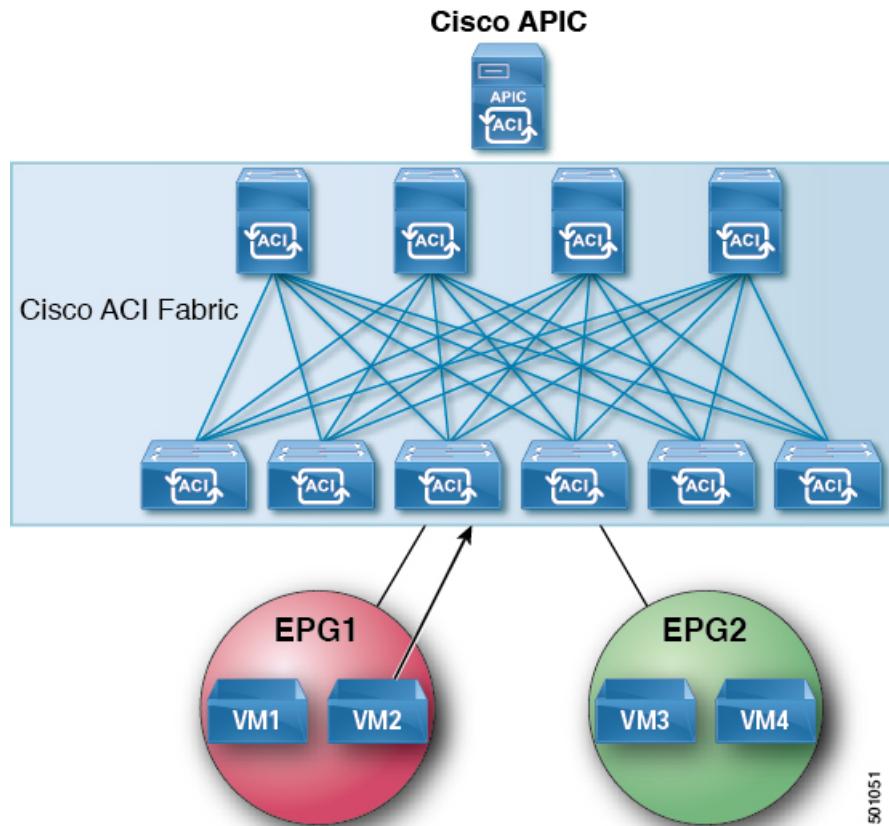


Table 7: ARP Table State

Device	State
VM1	IP = VM2 IP; MAC = ?
ACI fabric	IP = VM1 IP; MAC = VM1 MAC
VM2	IP = VM1 IP; MAC = BD MAC

- 5 VM2 is learned.

Figure 27: VM2 is Learned

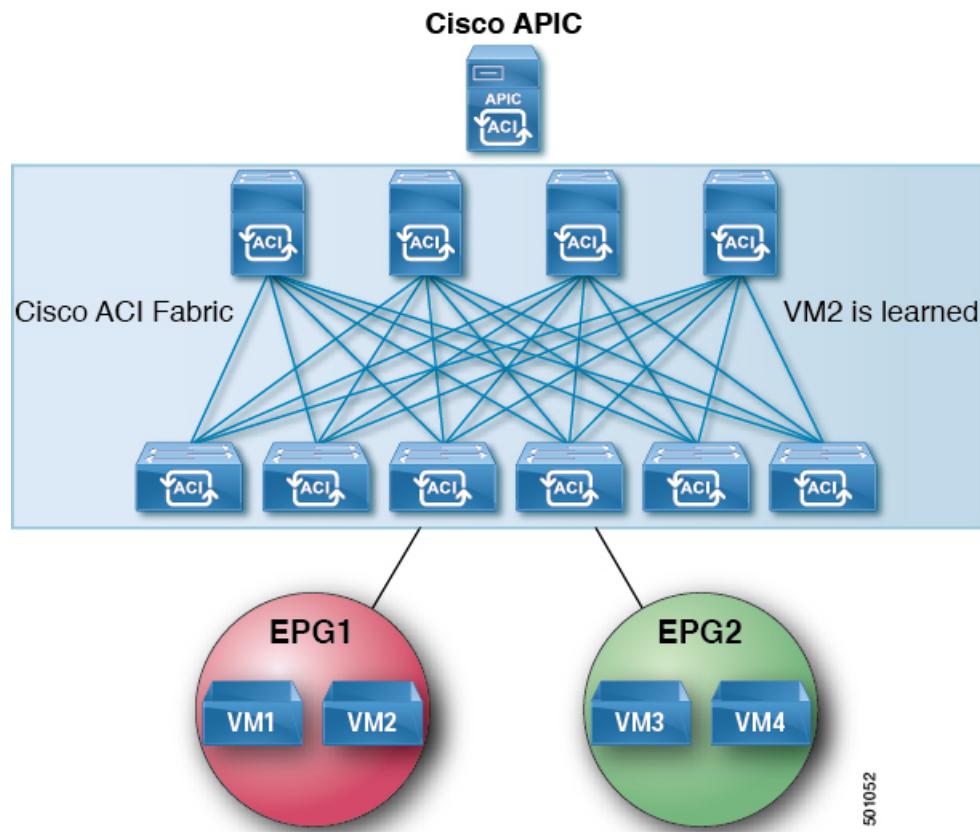


Table 8: ARP Table State

Device	State
VM1	IP = VM2 IP; MAC = ?
ACI fabric	IP = VM1 IP; MAC = VM1 MAC IP = VM2 IP; MAC = VM2 MAC
VM2	IP = VM1 IP; MAC = BD MAC

- 6 VM1 sends an ARP request with a broadcast MAC address to VM2.

Figure 28: VM1 Sends an ARP Request with a Broadcast MAC Address to VM2

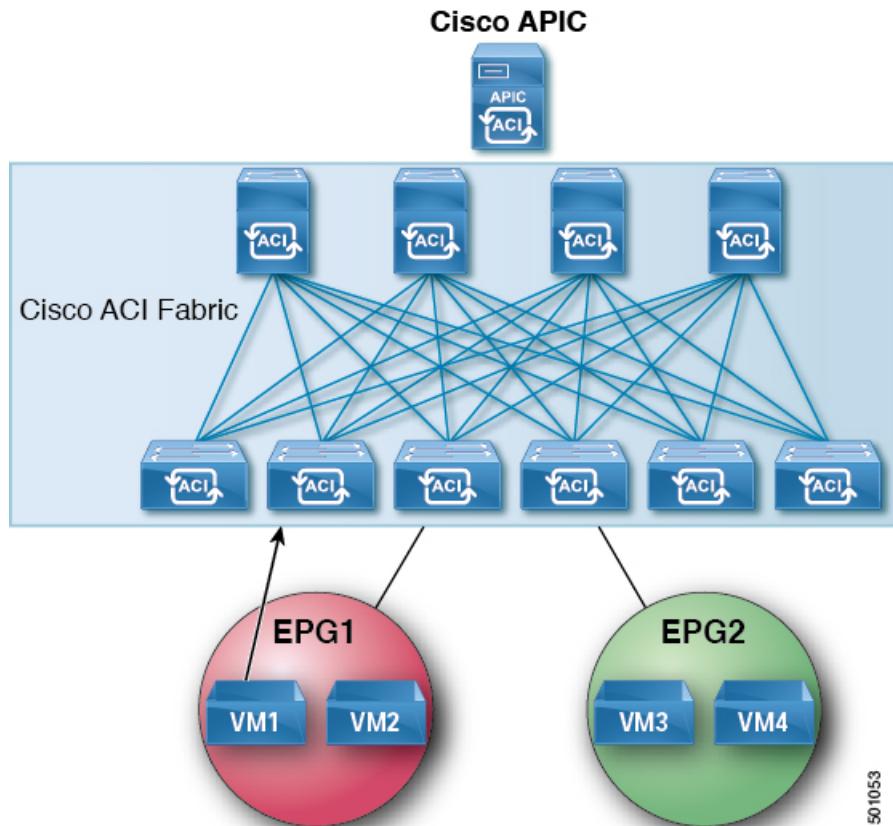


Table 9: ARP Table State

Device	State
VM1	IP = VM2 IP MAC = ?
ACI fabric	IP = VM1 IP; MAC = VM1 MAC IP = VM2 IP; MAC = VM2 MAC
VM2	IP = VM1 IP; MAC = BD MAC

- The ACI fabric sends a proxy ARP response to VM1.

Figure 29: ACI Fabric Sends a Proxy ARP Response to VM1

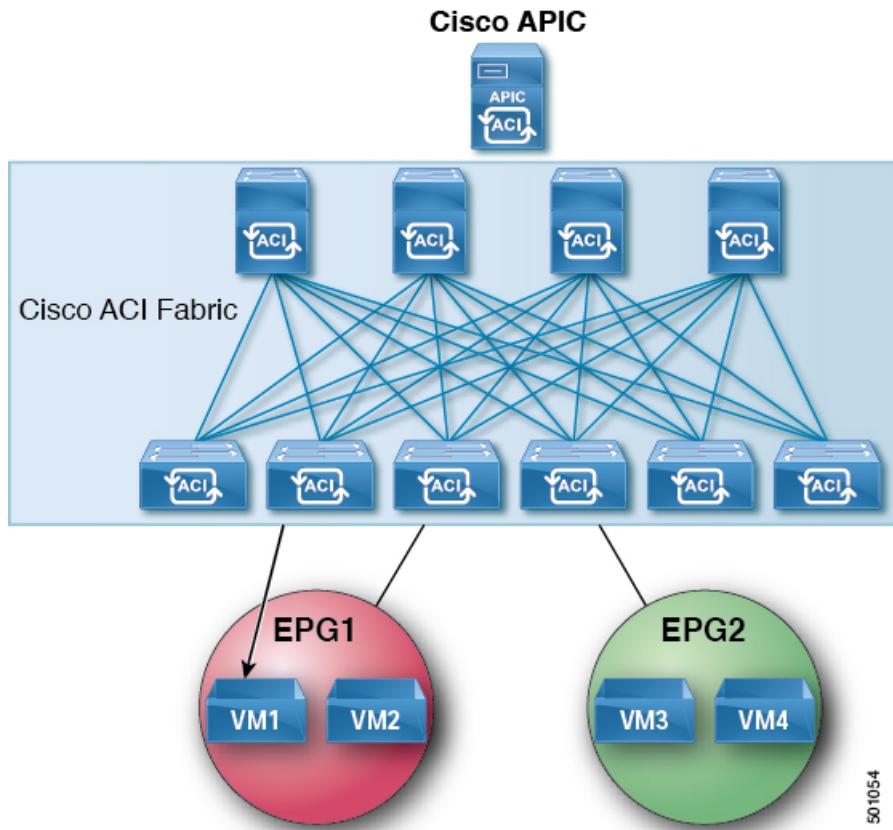


Table 10: ARP Table State

Device	State
VM1	IP = VM2 IP; MAC = BD MAC
ACI fabric	IP = VM1 IP; MAC = VM1 MAC IP = VM2 IP; MAC = VM2 MAC
VM2	IP = VM1 IP; MAC = BD MAC

Guidelines and Limitations

Consider these guidelines and limitations when using Proxy ARP:

- Proxy ARP is supported only on isolated EPGs. If an EPG is not isolated, a fault will be raised. For communication to happen within isolated EPGs with proxy ARP enabled, you must configure uSeg EPGs. For example, within the isolated EPG, there could be multiple VMs with different IP addresses, and you can configure a uSeg EPG with IP attributes matching the IP address range of these VMs.
- ARP requests from isolated endpoints to regular endpoints and from regular endpoints to isolated endpoints do not use proxy ARP. In such cases, endpoints communicate using the real MAC addresses of destination VMs.

Configuring Proxy ARP Using the REST API

Before You Begin

- Intra-EPG isolation must be enabled on the EPG where proxy ARP has to be enabled.

Procedure

Configure proxy ARP.

Example:

```
<polUni>
  <fvTenant name="Tenant1" status="">
    <fvCtx name="EngNet"/>
    <!-- bridge domain -->
    <fvBD name="BD1">
      <fvRsCtx tnFvCtxName="EngNet" />
      <fvSubnet ip="1.1.1.1/24"/>
    </fvBD>
    <fvAp name="Tenant1_app">
      <fvAEPg name="Tenant1_epg" pcEnfPref="enforced" fwdCtrl="proxy-arp">
        <fvRsBd tnFvBDName="BD1" />
        <fvRsDomAtt tDn="uni/vmmp-VMware/dom-dom9"/>
      </fvAEPg>
    </fvAp>
  </fvTenant>
</polUni>
```




CHAPTER 13

Provisioning Layer 3 Outside Connections

- [Layer 3 Outside Connections, page 229](#)
- [Route Controls, page 232](#)
- [BGP EVPN Services for Fabric WAN, page 234](#)
- [Multipod, page 242](#)
- [HSRP, page 247](#)
- [IP Multicast, page 250](#)
- [Pervasive Gateway, page 254](#)
- [Explicit Prefix Lists, page 255](#)
- [IP Address Aging Tracking, page 260](#)
- [Route Summarization, page 260](#)
- [External Route Interleak, page 264](#)
- [Routing Protocols, page 265](#)
- [Neighbor Discovery, page 276](#)

Layer 3 Outside Connections

Configuring a Tenant Layer 3 Outside Network Connection Overview

This topic provides a typical example of how to configure a Layer 3 Outside for tenant networks when using Cisco APIC.

Configuring Layer 3 Outside for Tenant Networks Using the REST API

The external routed network configured in the example can also be extended to support IPv4. Both IPv4 and IPv6 routes can be advertised to and learned from the external routed network.

Before You Begin

- The tenant, private network, and bridge domain are created.
- The external routed domain is created.

Procedure

Configure L3 Outside for tenant networks and associate the bridge domain with the Layer3 Outside.

Example:

```
<fvTenant name='TenantName'>
    <l3extOut name="L3Out1" enforceRtctrl="import,export">
        <l3extRsL3DomAtt tDn="uni/l3dom-l3DomP"/>
        <l3extLNodeP name="LNodeP1" >
            <l3extRsNodeL3OutAtt rtrId="1.2.3.4" tDn="topology/pod-1/node-101">
                <l3extLoopBackIfP addr="10.10.11.1" />
                <l3extLoopBackIfP addr="2000::3" />
            </l3extRsNodeL3OutAtt>
            <l3extLIfP name="IFP1" >
                <l3extRsPathL3OutAtt addr="10.11.12.10/24" ifInstT="l3-port" tDn="topology/pod-1/paths-103/pathep-[eth1/17]" />
            </l3extLIfP>
            <l3extLIfP name="IFP2" >
                <l3extRsPathL3OutAtt addr="2001::3/64" ifInstT="l3-port" tDn="topology/pod-1/paths-103/pathep-[eth1/17]" />
            </l3extLIfP>
        </l3extLNodeP>
        <l3extRsEctx tnFvCtxName="VRF1"/>
        <l3extInstP name="InstP1" >
            <l3extSubnet ip="192.168.1.0/24" scope="import-security" aggregate="" />
            <l3extSubnet ip="0.0.0.0/0" scope="export-rtctrl,import-rtctrl,import-security" aggregate="export-rtctrl,import-rtctrl" />
            <l3extSubnet ip="192.168.2.0/24" scope="export-rtctrl" aggregate="" />
            <l3extSubnet ip="::/0" scope="import-rtctrl,import-security" aggregate="import-rtctrl" />
            <l3extSubnet ip="2001:17a::/64" scope="export-rtctrl" aggregate="" />
        </l3extInstP>
    </l3extOut>
</fvTenant>
```

Note The "enforceRtctrl=import" is not applicable for EIGRP.

Tenant External Network Policy Example

The following XML code is an example of a Tenant Layer 3 external network policy.

```
<polUni>
    <fvTenant name='t0'>
        <fvCtx name="o1">
            <fvRsOspfCtxPol tnOspfCtxPolName="ospfCtxPol"/>
        </fvCtx>
        <fvCtx name="o2">
        </fvCtx>

        <fvBD name="bd1">
            <fvRsBDToOut tnl3extOutName='T0-o1-L3OUT-1' />
            <fvSubnet ip='10.16.1.1/24' scope='public' />
            <fvRsCtx tnFvCtxName="o1" />
        </fvBD>
    </fvTenant>
</polUni>
```

```

<fvAp name="AP1">
    <fvAEPg name="bd1-epg1">
        <fvRsCons tnVzBrCPName="vzBrCP-1">
        </fvRsCons>
        <fvRsProv tnVzBrCPName="vzBrCP-1">
        </fvRsProv>
        <fvSubnet ip='10.16.2.1/24' scope='private' />
        <fvSubnet ip='10.16.3.1/24' scope='private' />
        <fvRsBd tnFvBDName="bd1"/>
        <fvRsDomAtt tDn="uni/phys-physDomP"/>
        <fvRsPathAtt
            tDn="topology/pod-1/paths-101/pathep-[eth1/40]"
            encap='vlan-100'
            mode='regular'
            instrImedcy='immediate' />
    </fvAEPg>

    <fvAEPg name="bd1-epg2">
        <fvRsCons tnVzBrCPName="vzBrCP-1">
        </fvRsCons>
        <fvRsProv tnVzBrCPName="vzBrCP-1">
        </fvRsProv>
        <fvSubnet ip='10.16.4.1/24' scope='private' />
        <fvSubnet ip='10.16.5.1/24' scope='private' />
        <fvRsBd tnFvBDName="bd1"/>
        <fvRsDomAtt tDn="uni/phys-physDomP"/>
        <fvRsPathAtt
            tDn="topology/pod-1/paths-101/pathep-[eth1/41]"
            encap='vlan-200'
            mode='regular'
            instrImedcy='immediate' />
    </fvAEPg>
</fvAp>

<l3extOut name="T0-o1-L3OUT-1">
    <l3extRsEctx tnFvCtxName="o1"/>
    <ospfExtP areaId='60' />
    <l3extInstP name="l3extInstP-1">
        <fvRsCons tnVzBrCPName="vzBrCP-1">
        </fvRsCons>
        <fvRsProv tnVzBrCPName="vzBrCP-1">
        </fvRsProv>
        <l3extSubnet ip="192.5.1.0/24" />
        <l3extSubnet ip="192.5.2.0/24" />
        <l3extSubnet ip="192.6.0.0/16" />
        <l3extSubnet ip="199.0.0.0/8" />
    </l3extInstP>

    <l3extLNodeP name="l3extLNodeP-1">
        <l3extRsNodeL3OutAtt
            tDn="topology/pod-1/node-101" rtrId="10.17.1.1">
            <ipRouteP ip="10.16.101.1/32">
                <ipNexthopP nhAddr="10.17.1.99"/>
            </ipRouteP>
            <ipRouteP ip="10.16.102.1/32">
                <ipNexthopP nhAddr="10.17.1.99"/>
            </ipRouteP>
            <ipRouteP ip="10.17.1.3/32">
                <ipNexthopP nhAddr="10.11.2.2"/>
            </ipRouteP>
        </l3extRsNodeL3OutAtt >

        <l3extLIfP name='l3extLIfP-1'>
            <l3extRsPathL3OutAtt
                tDn="topology/pod-1/paths-101/pathep-[eth1/25]"
                encap='vlan-1001'
                ifInstT='sub-interface'
                addr="10.11.2.1/24"
                mtu="1500"/>
            <ospfIfP>
                <ospfRsIfPol tnOspfIfPolName='ospfIfPol' />

```

```

        </ospfIfP>
    </l3extLIfP>
</l3extLNodeP>
</l3extOut>

<ospfIfPol name="ospfIfPol" />
<ospfCtxPol name="ospfCtxPol" />

<vzFilter name="vzFilter-in-1">
    <vzEntry name="vzEntry-in-1"/>
</vzFilter>
<vzFilter name="vzFilter-out-1">
    <vzEntry name="vzEntry-out-1"/>
</vzFilter>

<vzBrCP name="vzBrCP-1">
    <vzSubj name="vzSubj-1">
        <vzInTerm>
            <vzRsFiltAtt tnVzFilterName="vzFilter-in-1"/>
        </vzInTerm>
        <vzOutTerm>
            <vzRsFiltAtt tnVzFilterName="vzFilter-out-1"/>
        </vzOutTerm>
    </vzSubj>
</vzBrCP>
</fvTenant>
</polUni>

```

Route Controls

Configuring a Routing Control Protocol Using Import and Export Controls

This topic provides a typical example that shows how to configure a routing control protocol using import and export controls. It assumes that you have configured Layer 3 outside network connections with BGP. You can also perform these tasks for a Layer 3 outside network configured with OSPF.

Configuring a Route Control Protocol to Use Import and Export Controls, With the REST API

This example assumes that you have configured the Layer 3 outside network connections using BGP. It is also possible to perform these tasks for a network using OSPF.

Before You Begin

- The tenant, private network, and bridge domain are created.
- The Layer 3 outside tenant network is configured.

Procedure

Configure the route control protocol using import and export controls.

Example:

```

<l3extOut descr="" dn="uni/tn-Ten_ND/out-L3Out1" enforceRtctrl="export" name="L3Out1"
ownerKey="" ownerTag="" targetDscp="unspecified">
    <l3extLNodeP descr="" name="LNodeP1" ownerKey="" ownerTag="" tag="yellow-green"
targetDscp="unspecified">
        <l3extRsNodeL3OutAtt rtrId="1.2.3.4" rtrIdLoopBack="yes"
tDn="topology/pod-1/node-101">

```

```

        <13extLoopBackIfP addr="2000::3" descr="" name="" />
    </13extRsNodeL3OutAtt>
    <13extLIfP descr="" name="IFP1" ownerKey="" ownerTag="" tag="yellow-green">
        <ospfIfP authKeyId="1" authType="none" descr="" name="" />
            <ospfRsIfPol tnOspfIfPolName="" />
        </ospfIfP>
        <13extRsNdIfPol tnNdIfPolName="" />
        <13extRsPathL3OutAtt addr="10.11.12.10/24" descr="" encaps="unknown"
ifInstT="13-port"
llAddr="::" mac="00:22:BD:F8:19:FF" mtu="1500" tDn="topology/pod-1/paths-101/pathep-[eth1/17]"
targetDscp="unspecified" />
        </13extLIfP>
    </13extLNodeP>
    <13extRsEctx tnFvCtxName="PVN1" />
    <13extInstP descr="" matchT="AtleastOne" name="InstP1" prio="unspecified"
targetDscp="unspecified" />
        <fvRsCustQosPol tnQosCustomPolName="" />
        <13extSubnet aggregate="" descr="" ip="192.168.1.0/24" name="" scope="" />
    </13extInstP>
    <ospfExtP areaCost="1" areaCtrl="redistribute,summary" areaId="0.0.0.1"
areaType="nssa" descr="" />
        <rtctrlProfile descr="" name="default-export" ownerKey="" ownerTag="" />
            <rtctrlCtxP descr="" name="routecontrolpvtnw" order="3" />
                <rtctrlScope descr="" name="" />
                    <rtctrlRsScopeToAttrP tnRtctrlAttrPName="actionruleprofile2" />
                </rtctrlScope>
            </rtctrlCtxP>
        </rtctrlProfile>
    </13extOut>
</13ext>

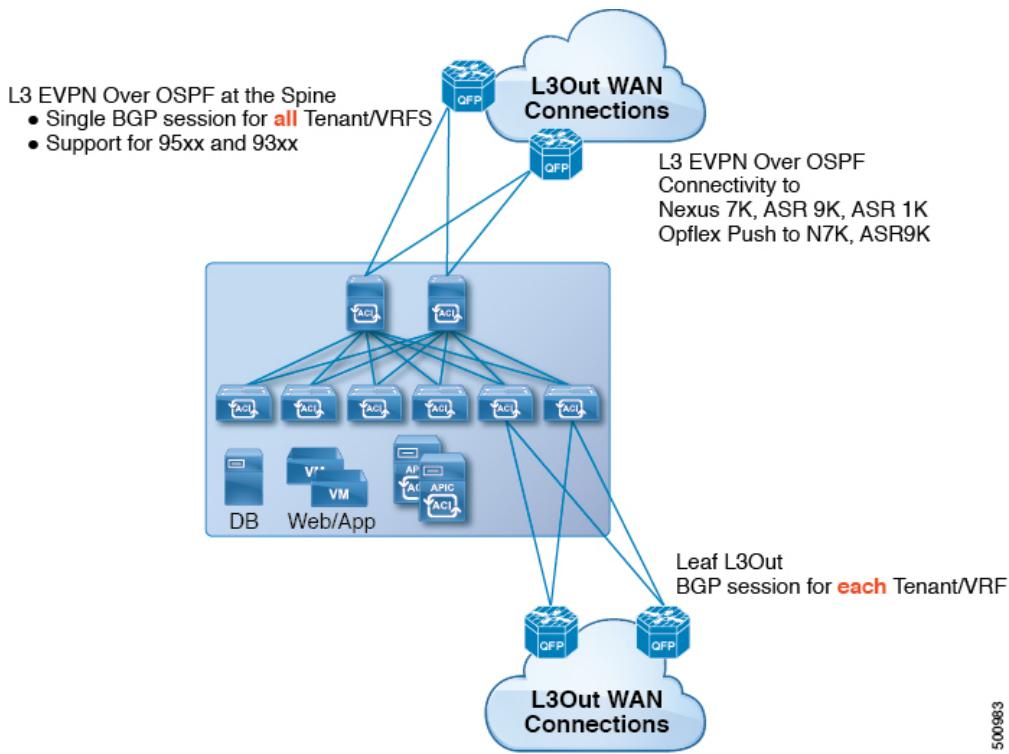
```

BGP EVPN Services for Fabric WAN

Layer 3 EVPN Services Over Fabric WAN

The Layer 3 EVPN services over fabric WAN feature enables much more efficient and scalable ACI fabric WAN connectivity. It uses the BGP EVPN protocol over OSPF for WAN routers that are connected to spine switches.

Figure 30: Layer 3 EVPN Services Over Fabric WAN



All tenant WAN connections use a single session on the spine switches where the WAN routers are connected. This aggregation of tenant BGP sessions towards the Data Center Interconnect Gateway (DCIG) improves control plane scale by reducing the number of tenant BGP sessions and the amount of configuration required for all of them. The network is extended out using L3-subinterfaces configured on spine fabric ports. Transit routing with shared services using Layer 3 EVPN services over fabric WAN is not supported.

A Layer 3 external outside network (`L3extOut`) for EVPN physical connectivity for a spine switch is specified under the `infra` tenant, and includes the following:

- `LNodeP` (`l3extInstP` is not required within the L3Out in Tenant Infra)
- A provider label for the `L3extOut` for EVPN in tenant infra.
- OSPF protocol policies
- BGP protocol policies

All regular tenants use the above-defined physical connectivity. The `L3extOut` defined in regular tenants only needs the following:

- An `l3extConsLbl` consumer label that must be matched with the same provider label of an `L3extOut` for EVPN in the `infra` tenant. Label matching enables application EPGs in other tenants to consume the `LNodeP` external `L3extOut` EPG.
- An `l3extInstP` with subnets and contracts. The scope of the subnet is used to control import/export route control and security policies.
- The BGP EVPN session in the matching provider `L3extOut` in the `infra` tenant advertises the tenant routes defined in this `L3extOut`.

Observe the following Layer 3 EVPN services for fabric WAN guidelines and limitations:

- At this time, only a single Layer 3 EVPN Services Over Fabric WAN provider policy can be deployed on spine switch interfaces for the whole fabric.
- Up to APIC release 2.0(2), Layer 3 EVPN is not supported with multipod. In release 2.0 (2) the two features are supported in the same fabric only over Cisco Nexus N9000K switches without "EX" on the end of the switch name; for example, N9K-9312TX. Since the 2.1(1) release, the two features can be deployed together over all the switches used in the multipod and EVPN topologies.
- When configuring Layer 3 EVPN connectivity on a spine switch, wait for the control plane to converge before configuring Layer 3 EVPN on another spine.
- A spine switch can be added to multiple provider Layer 3 EVPN Outs but the provider labels have to be different for each Layer 3 EVPN Out. Also, in this case, the OSPF Area has to be different on each of the `L3extOuts` and use different loopback addresses.
- The BGP EVPN session in the matching provider `L3extOut` in the `infra` tenant advertises the tenant routes defined in this `L3extOut`.
- When deploying three L3 EVPN Outs, if only 1 has a provider/consumer label for L3 EVPN, and 0/0 export aggregation, APIC will export all routes. This is the same as existing `L3extOut` on leaf switches for tenants.
- If there is direct peering between a spine switch and a data center interconnect (DCI) router, the transit routes from leaf switches to the ASR have the next hop as the PTEP of the leaf. In this case, define a static route on the ASR for the TEP range of that ACI pod. Also, if the DCI is dual-homed to the same pod, then the precedence (administrative distance) of the static route should be the same as the route received through the other link.
- The default `bgpPeerPfxPol` policy restricts routes to 20,000. For Layer 3 EVPN peers, increase this as needed.
- In a deployment scenario where there are two `L3extOuts` on one spine, and one of them has the provider label prov1 and peers with the DCI 1, the second `L3extOut` peers with DCI 2 with provider label prov2. If the tenant VRF has a consumer label pointing to any 1 of the provider labels (either prov1 or prov2), the tenant route will be sent out both DCI 1 and DCI 2.

Configuring Layer 3 EVPN for WAN Services Using the REST API

Procedure

- Step 1** The following example shows how to deploy nodes and spine switch interfaces for L3 EVPN for WAN services using the REST API:

Example:

POST
<https://192.0.20.123/api/mo/uni/golf.xml>

- Step 2** The XML below configures the spine switch interfaces and infra tenant provider of the Layer 3 EVPN WAN service. Include this XML structure in the body of the POST message.

Example:

```

<13extOut descr="" dn="uni/tn-infra/out-golf" enforceRtctrl="export,import"
    name="golf"
    ownerKey="" ownerTag="" targetDscp="unspecified">
<13extRsEctx tnFvCtxName="overlay-1"/>
<13extProvLbl descr="" name="golf">
    ownerKey="" ownerTag="" tag="yellow-green"/>
<13extLNodeP configIssues="" descr=""
    name="bLeaf" ownerKey="" ownerTag=""
    tag="yellow-green" targetDscp="unspecified">
    <13extRsNodeL3OutAtt rtrId="10.10.3.3" rtrIdLoopBack="no"
        tDn="topology/pod-1/node-111">
        <13extIntraNodeP descr="" fabricExtCtrlPeering="yes" name="" />
        <13extLoopBackIfP addr="10.10.3.3" descr="" name="" />
    </13extRsNodeL3OutAtt>
<13extRsNodeL3OutAtt rtrId="10.10.3.4" rtrIdLoopBack="no"
    tDn="topology/pod-1/node-112">
    <13extIntraNodeP descr="" fabricExtCtrlPeering="yes" name="" />
    <13extLoopBackIfP addr="10.10.3.4" descr="" name="" />
</13extRsNodeL3OutAtt>
<13extLifP descr="" name="portIf-spine1-3"
    ownerKey="" ownerTag="" tag="yellow-green">
        <ospfIfP authKeyId="1" authType="none" descr="" name="" />
        <ospfRsIfPol tnOspfIfPolName="ospfIfPol"/>
    </ospfIfP>
    <13extRsNdIfPol tnNdIfPolName="" />
    <13extRsIngressQosDppPol tnQosDppPolName="" />
    <13extRsEgressQosDppPol tnQosDppPolName="" />
    <13extRsPathL3OutAtt addr="7.2.1.1/24" descr="" 
        encapsulation="vlan-4"
        encapsScope="local"
        ifInstT="sub-interface"
        llAddr=":: mac="00:22:BD:F8:19:FF"
        mode="regular"
        mtu="1500"
        tDn="topology/pod-1/paths-111/pathEp-[eth1/12]"
        targetDscp="unspecified"/>
</13extLifP>
<13extLifP descr="" name="portIf-spine2-1"
    ownerKey=""
    ownerTag=""
    tag="yellow-green">
        <ospfIfP authKeyId="1"
            authType="none"
            descr=""
            name="" />
        <ospfRsIfPol tnOspfIfPolName="ospfIfPol"/>
    </ospfIfP>
    <13extRsNdIfPol tnNdIfPolName="" />
    <13extRsIngressQosDppPol tnQosDppPolName="" />

```

```

<13extRsEgressQosDppPol tnQosDppPolName="" />
<13extRsPathL3OutAtt addr="7.1.0.1/24" descr="" 
    encap="vlan-4"
    encapScope="local"
    ifInstT="sub-interface"
    llAddr="::" mac="00:22:BD:F8:19:FF"
    mode="regular"
    mtu="9000"
    tDn="topology/pod-1/paths-112/pathep-[eth1/11]"
    targetDscp="unspecified"/>
</13extLIfP>
<13extLIfP descr="" name="portif-spine2-2"
    ownerKey=""
    ownerTag=""
    tag="yellow-green">
    <ospfIfP authKeyId="1"
        authType="none" descr=""
        name="">
        <ospfRsIfPol tnOspfIfPolName="ospfIfPol"/>
    </ospfIfP>
    <13extRsNdIfPol tnNdIfPolName="" />
    <13extRsIngressQosDppPol tnQosDppPolName="" />
    <13extRsEgressQosDppPol tnQosDppPolName="" />
    <13extRsPathL3OutAtt addr="7.2.2.1/24" descr="" 
        encap="vlan-4"
        encapScope="local"
        ifInstT="sub-interface"
        llAddr="::" mac="00:22:BD:F8:19:FF"
        mode="regular"
        mtu="1500"
        tDn="topology/pod-1/paths-112/pathep-[eth1/12]"
        targetDscp="unspecified"/>
    </13extLIfP>
    <13extLIfP descr="" name="portIf-spine1-2"
        ownerKey="" ownerTag="" tag="yellow-green">
        <ospfIfP authKeyId="1" authType="none" descr="" name="">
            <ospfRsIfPol tnOspfIfPolName="ospfIfPol"/>
        </ospfIfP>
        <13extRsNdIfPol tnNdIfPolName="" />
        <13extRsIngressQosDppPol tnQosDppPolName="" />
        <13extRsEgressQosDppPol tnQosDppPolName="" />
        <13extRsPathL3OutAtt addr="9.0.0.1/24" descr="" 
            encap="vlan-4"
            encapScope="local"
            ifInstT="sub-interface"
            llAddr="::" mac="00:22:BD:F8:19:FF"
            mode="regular"
            mtu="9000"
            tDn="topology/pod-1/paths-111/pathep-[eth1/11]"
            targetDscp="unspecified"/>
        </13extLIfP>
        <13extLIfP descr="" name="portIf-spine1-1"
            ownerKey="" ownerTag="" tag="yellow-green">
            <ospfIfP authKeyId="1" authType="none" descr="" name="">
                <ospfRsIfPol tnOspfIfPolName="ospfIfPol"/>
            </ospfIfP>
            <13extRsNdIfPol tnNdIfPolName="" />
            <13extRsIngressQosDppPol tnQosDppPolName="" />
            <13extRsEgressQosDppPol tnQosDppPolName="" />
            <13extRsPathL3OutAtt addr="7.0.0.1/24" descr="" 
                encap="vlan-4"
                encapScope="local"
                ifInstT="sub-interface"
                llAddr="::" mac="00:22:BD:F8:19:FF"
                mode="regular"
                mtu="1500"
                tDn="topology/pod-1/paths-111/pathep-[eth1/10]"
                targetDscp="unspecified"/>
            </13extLIfP>
            <bpgIntraPeerP addr="10.10.3.2"
                allowedSelfAsCnt="3"
                ctrl="send-com, send-ext-com">

```

```

        descr="" name="" peerCtrl=""
peerT="wan"
privateASctrl="" ttl="2" weight="0">
<bgpRsPeerPfxPol tnBgpPeerPfxPolName="" />
<bpgAsP asn="150" descr="" name="aspn" />
</bgpInfraPeerP>
<bgpInfraPeerP addr="10.10.4.1"
allowedSelfAsCnt="3"
ctrl="send-com, send-ext-com" descr="" name="" peerCtrl=""
peerT="wan"
privateASctrl="" ttl="1" weight="0">
<bgpRsPeerPfxPol tnBgpPeerPfxPolName="" />
<bpgAsP asn="100" descr="" name="" />
</bgpInfraPeerP>
<bgpInfraPeerP addr="10.10.3.1"
allowedSelfAsCnt="3"
ctrl="send-com, send-ext-com" descr="" name="" peerCtrl=""
peerT="wan"
privateASctrl="" ttl="1" weight="0">
<bgpRsPeerPfxPol tnBgpPeerPfxPolName="" />
<bpgAsP asn="100" descr="" name="" />
</bgpInfraPeerP>
</l3extLNodeP>
<bgpRtTargetInstrP descr="" name="" ownerKey="" ownerTag="" rtTargetT="explicit"/>
<l3extRsL3DomAtt tDn="uni/l3dom-l3dom" />
<l3extInstP descr="" matchT="AtleastOne" name="golfInstP"
prio="unspecified"
targetDscp="unspecified">
<fvRsCustQosPol tnQosCustomPolName="" />
</l3extInstP>
<bgpExtP descr="" />
<ospfExtP areaCost="1"
areaCtrl="redistribute, summary"
areaId="0.0.0.1"
areaType="regular" descr="" />
</l3extOut>

```

Step 3 The XML below configures the tenant consumer of the infra Layer 3 EVPN WAN service. Include this XML structure in the body of the POST message.

Example:

```

<fvTenant descr="" dn="uni/tn-pep6" name="pep6" ownerKey="" ownerTag="" >
<vzBrCP descr="" name="webCtrct"
ownerKey="" ownerTag="" prio="unspecified"
scope="global" targetDscp="unspecified">
<vzSubj consMatchT="AtleastOne" descr=""
name="http" prio="unspecified" provMatchT="AtleastOne"
revFltPorts="yes" targetDscp="unspecified">
<vzRsSubjFiltAtt directives="" tnVzFilterName="default" />
</vzSubj>
</vzBrCP>
<vzBrCP descr="" name="webCtrct-pod2"
ownerKey="" ownerTag="" prio="unspecified"
scope="global" targetDscp="unspecified">
<vzSubj consMatchT="AtleastOne" descr=""
name="http" prio="unspecified"
provMatchT="AtleastOne" revFltPorts="yes"
targetDscp="unspecified">
<vzRsSubjFiltAtt directives="" tnVzFilterName="default" />
</vzSubj>
</vzBrCP>
<fvCtx descr="" knwMcastAct="permit"
name="ctx6" ownerKey="" ownerTag="" >
pcEnfDir="ingress" pcEnfPref="enforced">
<bgpRtTargetP af="ipv6-ucast"
descr="" name="" ownerKey="" ownerTag="" >
<bgpRtTarget descr="" name="" ownerKey="" ownerTag="" >
rt="route-target:as4-nn2:100:1256"
type="export" />

```

```

<bgpRtTarget descr="" name="" ownerKey="" ownerTag=""
    rt="route-target:as4-nn2:100:1256"
    type="import"/>
</bgpRtTargetP>
<bgpRtTargetP af="ipv4-ucast"
    descr="" name="" ownerKey="" ownerTag=""
    rt="route-target:as4-nn2:100:1256"
    type="export"/>
<bgpRtTarget descr="" name="" ownerKey="" ownerTag=""
    rt="route-target:as4-nn2:100:1256"
    type="import"/>
</bgpRtTargetP>
<fvRsCtxToExtRouteTagPol tnL3extRouteTagPolName="" />
<fvRsBgpCtxPol tnBgpCtxPolName="" />
<vzAny descr="" matchT="AtleastOne" name="" />
<fvRsOspfCtxPol tnOspfCtxPolName="" />
<fvRsCtxToEpRet tnFvEpRetPolName="" />
<l3extGlobalCtxName descr="" name="dci-pep6"/>
</fvCtx>
<fvBD arpFlood="no" descr="" epMoveDetectMode=""
    ipLearning="yes"
    limitIpLearnToSubnets="no"
    llAddr="::" mac="00:22:BD:F8:19:FF"
    mcastAllow="no"
    multiDstPktAct="bd-flood"
    name="bd107" ownerKey="" ownerTag="" type="regular"
    unicastRoute="yes"
    unkMacUcastAct="proxy"
    unkMcastAct="flood"
    vmac="not-applicable">
    <fvRsBDToNdP tnNdIfPolName="" />
    <fvRsBDToOut tnL3extOutName="routAccounting-pod2" />
    <fvRsCtx tnFvCtxName="ctx6" />
    <fvRsIgmpsn tnIgmpSnoopPolName="" />
    <fvSubnet ctrl="" descr="" ip="27.6.1.1/24"
        name="" preferred="no"
        scope="public"
        virtual="no" />
        <fvSubnet ctrl="nd" descr="" ip="2001:27:6:1::1/64"
            name="" preferred="no"
            scope="public"
            virtual="no" />
            <fvRsNdPfxPol tnNdPfxPolName="" />
        </fvSubnet>
        <fvRsBdToEpRet resolveAct="resolve" tnFvEpRetPolName="" />
    </fvSubnet>
    <fvRsBdToEpRet resolveAct="resolve" tnFvEpRetPolName="" />
</fvBD>
<fvBD arpFlood="no" descr="" epMoveDetectMode=""
    ipLearning="yes"
    limitIpLearnToSubnets="no"
    llAddr="::" mac="00:22:BD:F8:19:FF"
    mcastAllow="no"
    multiDstPktAct="bd-flood"
    name="bd103" ownerKey="" ownerTag="" type="regular"
    unicastRoute="yes"
    unkMacUcastAct="proxy"
    unkMcastAct="flood"
    vmac="not-applicable">
    <fvRsBDToNdP tnNdIfPolName="" />
    <fvRsBDToOut tnL3extOutName="routAccounting" />
    <fvRsCtx tnFvCtxName="ctx6" />
    <fvRsIgmpsn tnIgmpSnoopPolName="" />
    <fvSubnet ctrl="" descr="" ip="23.6.1.1/24"
        name="" preferred="no"
        scope="public"
        virtual="no" />
        <fvSubnet ctrl="nd" descr="" ip="2001:23:6:1::1/64"
            name="" preferred="no"
            scope="public" virtual="no" />
            <fvRsNdPfxPol tnNdPfxPolName="" />
        </fvSubnet>
        <fvRsBdToEpRet resolveAct="resolve" tnFvEpRetPolName="" />
    </fvSubnet>
    <fvRsBdToEpRet resolveAct="resolve" tnFvEpRetPolName="" />
</fvBD>

```

```

</fvBD>
<cvnsSvcCont/>
<fvRsTenantMonPol tnMonEPGPolName="" />
<fvAp descr="" name="AP1"
      ownerKey="" ownerTag="" prio="unspecified">
    <fvAEPg descr=""
            isAttrBasedEPg="no"
            matchT="AtleastOne"
            name="epg107"
            pcEnfPref="unenforced" prio="unspecified">
      <fvRsCons prio="unspecified"
                  tnVzBrCPName="webCtrct-pod2"/>
      <fvRsPathAtt descr=""
                    encaps="vlan-1256"
                    instrImedcy="immediate"
                    mode="regular" primaryEncap="unknown"
                    tDn="topology/pod-2/paths-107/pathep-[eth1/48]"/>
      <fvRsDomAtt classPref="encap" delimiter=""
                    encaps="unknown"
                    instrImedcy="immediate"
                    primaryEncap="unknown"
                    resImedcy="lazy" tDn="uni/phys-phys"/>
      <fvRsCustQosPol tnQosCustomPolName="" />
      <fvRsBd tnFvBDName="bd107"/>
      <fvRsProv matchT="AtleastOne"
                  prio="unspecified"
                  tnVzBrCPName="default"/>
    </fvAEPg>
    <fvAEPg descr=""
            isAttrBasedEPg="no"
            matchT="AtleastOne"
            name="epg103"
            pcEnfPref="unenforced" prio="unspecified">
      <fvRsCons prio="unspecified" tnVzBrCPName="default"/>
      <fvRsCons prio="unspecified" tnVzBrCPName="webCtrct"/>
      <fvRsPathAtt descr="" encaps="vlan-1256"
                    instrImedcy="immediate"
                    mode="regular" primaryEncap="unknown"
                    tDn="topology/pod-1/paths-103/pathep-[eth1/48]"/>
      <fvRsDomAtt classPref="encap" delimiter=""
                    encaps="unknown"
                    instrImedcy="immediate"
                    primaryEncap="unknown"
                    resImedcy="lazy" tDn="uni/phys-phys"/>
      <fvRsCustQosPol tnQosCustomPolName="" />
      <fvRsBd tnFvBDName="bd103"/>
    </fvAEPg>
  </fvAp>
<l3extOut descr=""
            enforceRtctrl="export"
            name="routAccounting-pod2"
            ownerKey="" ownerTag="" targetDscp="unspecified">
  <l3extRsEctx tnFvCtxName="ctx6"/>
  <l3extInstP descr=""
                matchT="AtleastOne"
                name="accountingInst-pod2"
                prio="unspecified" targetDscp="unspecified">
    <l3extSubnet aggregate="export-rtctrl,import-rtctrl"
                  descr="" ip="::/0" name=""
                  scope="export-rtctrl,import-rtctrl,import-security"/>
    <l3extSubnet aggregate="export-rtctrl,import-rtctrl"
                  descr=""
                  ip="0.0.0.0/0" name=""
                  scope="export-rtctrl,import-rtctrl,import-security"/>
  <fvRsCustQosPol tnQosCustomPolName="" />
  <fvRsProv matchT="AtleastOne"
                  prio="unspecified" tnVzBrCPName="webCtrct-pod2"/>
</l3extInstP>
<b><l3extConsLbl descr="" name="golf2" owner="infra" ownerKey="" ownerTag="" tag="yellow-green"/></b>

```

```

</l3extOut>
<l3extOut descr="">
  enforceRtctrl="export"
  name="routAccounting"
  ownerKey=""
  ownerTag=""
  targetDscp="unspecified">
<l3extRsEctx tnFvCtxName="ctx6"/>
<l3extInstP descr="">
  matchT="AtleastOne"
  name="accountingInst"
  prio="unspecified" targetDscp="unspecified">
<l3extSubnet aggregate="export-rtctrl,import-rtctrl" descr="">
  ip="0.0.0.0/0" name=""
  scope="export-rtctrl,import-rtctrl,import-security"/>
<fvRsCustQosPol tnQosCustomPolName="" />
<fvRsProv matchT="AtleastOne" prio="unspecified" tnVzBrCPName="webCtrct" />
</l3extInstP>
<l3extConsLbl descr="">
  name="golf"
  owner="infra"
  ownerKey="" ownerTag="" tag="yellow-green"/>
</l3extOut>
</fvTenant>

```

Distributing BGP EVPN Type-2 Host Routes to a DCIG

In APIC up to release 2.0(1f), the fabric control plane did not send EVPN host routes directly, but advertised public bridge domain (BD) subnets in the form of BGP EVPN type-5 (IP Prefix) routes to a Data Center Interconnect Gateway (DCIG). This could result in suboptimal traffic forwarding. To improve forwarding, in APIC release 2.1x, you can enable fabric spines to also advertise host routes using EVPN type-2 (MAC-IP) host routes to the DCIG along with the public BD subnets.

To do so, you must perform the following steps:

- 1 When you configure the BGP Address Family Context Policy, enable Host Route Leak.
- 2 When you configure VRF properties:
 - a Add the BGP Address Family Context Policy to the BGP Context Per Address Families for IPv4 and IPv6.
 - b Configure BGP Route Target Profiles that identify routes that can be imported or exported from the VRF.

Enabling Distributing BGP EVPN Type-2 Host Routes to a DCIG Using the REST API

Enable distributing BGP EVPN type-2 host routes using the REST API, as follows:

Before You Begin

EVPN services must be configured.

Procedure

-
- Step 1** Configure the Host Route Leak policy, with a POST containing XML such as in the following example:

Example:

```
<bgpCtxAfPol descr="" ctrl="host-rt-leak" name="bgpCtxPol_0" status="" />
```

- Step 2** Apply the policy to the VRF BGP Address Family Context Policy for one or both of the address families using a POST containing XML such as in the following example:

Example:

```
<fvCtx name="vni-10001">
<fvRsCtxToBgpCtxAfPol af="ipv4-ucast" tnBgpCtxAfPolName="bgpCtxPol_0"/>
<fvRsCtxToBgpCtxAfPol af="ipv6-ucast" tnBgpCtxAfPolName="bgpCtxPol_0"/>
</fvCtx>
```

Multipod

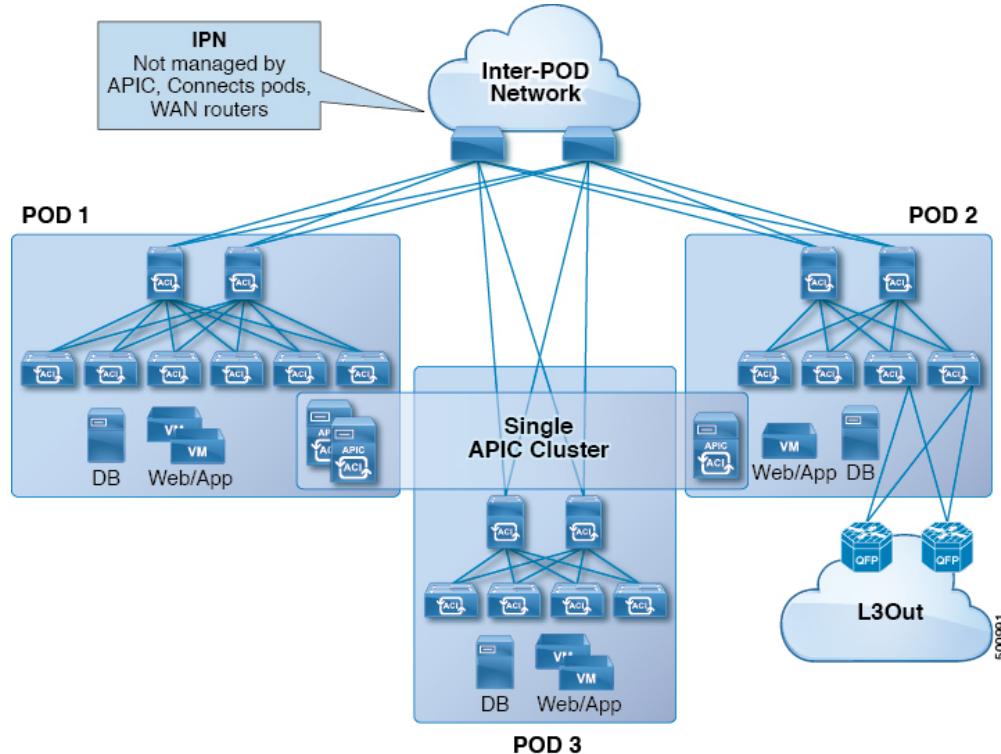
Multipod

Multipod enables provisioning a more fault tolerant fabric comprised of multiple pods with isolated control plane protocols. Also, multipod provides more flexibility with regard to the full mesh cabling between leaf and spine switches. For example, if leaf switches are spread across different floors or different buildings, multipod enables provisioning multiple pods per floor or building and providing connectivity between pods through spine switches.

Multipod uses MP-BGP EVPN as the control-plane communication protocol between the ACI spines in different Pods. WAN routers can be provisioned in the IPN, directly connected to spine switches, or connected

to border leaf switches. Multipod uses a single APIC cluster for all the pods; all the pods act as a single fabric. Individual APIC controllers are placed across the pods but they are all part of a single APIC cluster.

Figure 31: Multipod Overview



For control plane isolation, IS-IS and COOP are not extended across pods. Endpoints synchronize across pods using BGP EVPN over the IPN between the pods. Two spines in each pod are configured to have BGP EVPN sessions with spines of other pods. The spines connected to the IPN get the endpoints and multicast groups from COOP within a pod, but they advertise them over the IPN EVPN sessions between the pods. On the receiving side, BGP gives them back to COOP and COOP synchs them across all the spines in the pod. WAN routes are exchanged between the pods using BGP VPv4/VPv6 address families; they are not exchanged using the EVPN address family.

There are two modes of setting up the spine switches for communicating across pods as peers and route reflectors:

- **Automatic**

- Automatic mode is a route reflector based mode that does not support a full mesh where all spines peer with each other. The administrator must post an existing BGP route reflector policy and select interpod aware (EVPN) route reflectors. All the peer/client settings are automated by the APIC.
- The administrator does not have an option to choose route reflectors that don't belong to the fabric (for example, in the IPN).

- **Manual**

- The administrator has the option to configure full mesh where all spines peer with each other without route reflectors.
- In manual mode, the administrator must post the already existing BGP peer policy.

Observe the following multipod guidelines and limitations:

- When adding a pod to the ACI fabric, wait for the control plane to converge before adding another pod.
- OSPF is deployed on ACI spine switches and IPN switches to provide reachability between PODs. L3 subinterfaces are created on spines to connect to IPN switches. OSPF is enabled on these L3 subinterfaces and per POD TEP prefixes are advertised over OSPF. There is one subinterface created on each external spine link. Provision many external links on each spine if the expectation is that the amount of east-west traffic between PODs will be large. Currently, ACI spine switches support up to 64 external links on each spine, and each subinterface can be configured for OSPF. Spine proxy TEP addresses are advertised in OSPF over all the subinterfaces leading to a maximum of 64 way ECMP on the IPN switch for proxy TEP addresses. Similarly, spines would receive proxy TEP addresses of other PODs from IPN switches over OSPF and the spine can have up to 64 way ECMP for remote pod proxy TEP addresses. In this way, traffic between PODs spread over all these external links provides the desired bandwidth.
- When all fabric links of a spine switch are down, OSPF advertises the TEP routes with the maximum metric. This will force the IPN switch to remove the spine switch from ECMP which will prevent the IPN from forwarding traffic to the down spine switch. Traffic will be received by other spines that have up fabric links.
- Up to APIC release 2.0(2), multipod is not supported with Layer 3 EVPN. In release 2.0 (2) the two features are supported in the same fabric only over Cisco Nexus N9000K switches without "EX" on the end of the switch name; for example, N9K-9312TX. Since the 2.1(1) release, the two features can be deployed together over all the switches used in the multipod and EVPN topologies.
- In a multipod fabric, if a spine in POD1 uses the infra tenant L3extOut-1, the TORs for the other pods (POD2, POD3) cannot use the same infra L3extOut (L3extOut-1) for Layer 3 EVPN control plane connectivity. Each POD must use their own spine switch and infra L3extOut.
- No filtering is done for limiting the routes exchanged across pods. All end-point and WAN routes present in each pod are exported to other pods.
- Inband management across pods is automatically configured by a self tunnel on every spine.

Setting Up Multipod Fabric Using the REST API

Procedure

Step 1 Login:

Example:

```
http://<apic-name/ip>:80/api/aaaLogin.xml  
data: <aaaUser name="admin" pwd="ins3965!"/>
```

Step 2 Configure the TEP pool:

Example:

```
http://<apic-name/ip>:80/api/policymgr/mo/uni/controller.xml
```

```
<fabricSetupPol status=''>
    <fabricSetupP podId="1" tepPool="10.0.0.0/16" />
    <fabricSetupP podId="2" tepPool="10.1.0.0/16" status=' ' />
</fabricSetupPol>
```

Step 3 Configure the node ID policy:**Example:**

```
http://<apic-name/ip>:80/api/node/mo/uni/controller.xml
```

```
<fabricNodeIdentPol>
<fabricNodeIdentP serial="SAL1819RXP4" name="ifav4-leaf1" nodeId="101" podId="1"/>
<fabricNodeIdentP serial="SAL1803L25H" name="ifav4-leaf2" nodeId="102" podId="1"/>
<fabricNodeIdentP serial="SAL1934MNY0" name="ifav4-leaf3" nodeId="103" podId="1"/>
<fabricNodeIdentP serial="SAL1934MNY3" name="ifav4-leaf4" nodeId="104" podId="1"/>
<fabricNodeIdentP serial="SAL1748H56D" name="ifav4-spine1" nodeId="201" podId="1"/>
<fabricNodeIdentP serial="SAL1938P7A6" name="ifav4-spine3" nodeId="202" podId="1"/>
<fabricNodeIdentP serial="SAL1938PHBB" name="ifav4-leaf5" nodeId="105" podId="2"/>
<fabricNodeIdentP serial="SAL1942R857" name="ifav4-leaf6" nodeId="106" podId="2"/>
<fabricNodeIdentP serial="SAL1931LA3B" name="ifav4-spine2" nodeId="203" podId="2"/>
<fabricNodeIdentP serial="FGE173400A9" name="ifav4-spine4" nodeId="204" podId="2"/>
</fabricNodeIdentPol>
```

Step 4 Configure infra L3Out and external connectivity profile:**Example:**

```
http://<apic-name/ip>:80/api/node/mo/uni.xml
```

```
<polUni>

<fvTenant descr="" dn="uni/tn-infra" name="infra" ownerKey="" ownerTag="">

    <l3extOut descr="" enforceRtctrl="export" name="multipod" ownerKey="" ownerTag="">
        targetDscp="unspecified" status=''
        <ospfExtP arealId='0' areaType='regular' status=''/>
        <bgpExtP status=' '/>
        <l3extRsEctx tnFvCtxName="overlay-1"/>
        <l3extProvLbl descr="" name="prov_mp1" ownerKey="" ownerTag="" tag="yellow-green"/>

        <l3extLNodeP name="bSpine">
            <l3extRsNodeL3OutAtt rtrId="201.201.201.201" rtrIdLoopBack="no"
tDn="topology/pod-1/node-201">
                <l3extIntraNodeP descr="" fabricExtCtrlPeering="yes" name="" />
                <l3extLoopBackIfP addr="201::201/128" descr="" name="" />
                <l3extLoopBackIfP addr="201.201.201.201/32" descr="" name="" />
            </l3extRsNodeL3OutAtt>

            <l3extRsNodeL3OutAtt rtrId="202.202.202.202" rtrIdLoopBack="no"
tDn="topology/pod-1/node-202">
                <l3extIntraNodeP descr="" fabricExtCtrlPeering="yes" name="" />
                <l3extLoopBackIfP addr="202::202/128" descr="" name="" />
                <l3extLoopBackIfP addr="202.202.202.202/32" descr="" name="" />
            </l3extRsNodeL3OutAtt>

            <l3extRsNodeL3OutAtt rtrId="203.203.203.203" rtrIdLoopBack="no"
tDn="topology/pod-2/node-203">
                <l3extIntraNodeP descr="" fabricExtCtrlPeering="yes" name="" />
                <l3extLoopBackIfP addr="203::203/128" descr="" name="" />
                <l3extLoopBackIfP addr="203.203.203.203/32" descr="" name="" />
            </l3extRsNodeL3OutAtt>

            <l3extRsNodeL3OutAtt rtrId="204.204.204.204" rtrIdLoopBack="no"
tDn="topology/pod-2/node-204">
                <l3extIntraNodeP descr="" fabricExtCtrlPeering="yes" name="" />
                <l3extLoopBackIfP addr="204::204/128" descr="" name="" />
            </l3extRsNodeL3OutAtt>
        </l3extLNodeP>
    </l3extOut>
</fvTenant>
```

```

        <13extLoopBackIfP addr="204.204.204.204/32" descr="" name="" />
    </13extRsNodeL3OutAtt>

    <13extLIfP name='portIf'>
        <13extRsPathL3OutAtt descr='asr' tDn="topology/pod-1/paths-201/pathEP-[eth1/1]"
encap='vlan-4' ifInstT='sub-interface' addr="201.1.1.1/30" />
        <13extRsPathL3OutAtt descr='asr' tDn="topology/pod-1/paths-201/pathEP-[eth1/2]"
encap='vlan-4' ifInstT='sub-interface' addr="201.2.1.1/30" />
        <13extRsPathL3OutAtt descr='asr' tDn="topology/pod-1/paths-202/pathEP-[eth1/2]"
encap='vlan-4' ifInstT='sub-interface' addr="202.1.1.1/30" />
        <13extRsPathL3OutAtt descr='asr' tDn="topology/pod-2/paths-203/pathEP-[eth1/1]"
encap='vlan-4' ifInstT='sub-interface' addr="203.1.1.1/30" />
        <13extRsPathL3OutAtt descr='asr' tDn="topology/pod-2/paths-203/pathEP-[eth1/2]"
encap='vlan-4' ifInstT='sub-interface' addr="203.2.1.1/30" />
        <13extRsPathL3OutAtt descr='asr' tDn="topology/pod-2/paths-204/pathEP-[eth4/31]"
encap='vlan-4' ifInstT='sub-interface' addr="204.1.1.1/30" />

    <ospfIfP>
        <ospfRsIfPol tnOspfIfPolName='ospfIfPol' />
    </ospfIfP>

    </13extLIfP>
</13extLNodeP>

    <13extInstP descr="" matchT="AtleastOne" name="instpl" prio="unspecified"
targetDscp="unspecified">
        <fvRsCustQosPol tnQosCustomPolName="" />
    </13extInstP>
</13extOut>

<fvFabricExtConnP descr="" id="1" name="Fabric_Ext_Conn_Poll" rt="extended:as2-nn4:5:16"
status=''>
    <fvPodConnP descr="" id="1" name="">
        <fvIp addr="100.11.1.1/32" />
    </fvPodConnP>
    <fvPodConnP descr="" id="2" name="">
        <fvIp addr="200.11.1.1/32" />
    </fvPodConnP>
    <fvPeeringP descr="" name="" ownerKey="" ownerTag="" type="automatic_with_full_mesh" />
    <13extFabricExtRoutingP descr="" name="ext_routing_prof_1" ownerKey="" ownerTag="">
        <13extSubnet aggregate="" descr="" ip="100.0.0.0/8" name="" scope="import-security" />
        <13extSubnet aggregate="" descr="" ip="200.0.0.0/8" name="" scope="import-security" />
        <13extSubnet aggregate="" descr="" ip="201.1.0.0/16" name="" scope="import-security" />
        <13extSubnet aggregate="" descr="" ip="201.2.0.0/16" name="" scope="import-security" />
        <13extSubnet aggregate="" descr="" ip="202.1.0.0/16" name="" scope="import-security" />
        <13extSubnet aggregate="" descr="" ip="203.1.0.0/16" name="" scope="import-security" />
        <13extSubnet aggregate="" descr="" ip="203.2.0.0/16" name="" scope="import-security" />
        <13extSubnet aggregate="" descr="" ip="204.1.0.0/16" name="" scope="import-security" />
    </13extFabricExtRoutingP>
    </fvFabricExtConnP>
</fvTenant>
</polUni>

```

HSRP

About HSRP

HSRP is a first-hop redundancy protocol (FHRP) that allows a transparent failover of the first-hop IP router. HSRP provides first-hop routing redundancy for IP hosts on Ethernet networks configured with a default router IP address. You use HSRP in a group of routers for selecting an active router and a standby router. In a group of routers, the active router is the router that routes packets, and the standby router is the router that takes over when the active router fails or when preset conditions are met.

Many host implementations do not support any dynamic router discovery mechanisms but can be configured with a default router. Running a dynamic router discovery mechanism on every host is not practical for many reasons, including administrative overhead, processing overhead, and security issues. HSRP provides failover services to such hosts.

When you use HSRP, you configure the HSRP virtual IP address as the default router of the host (instead of the IP address of the actual router). The virtual IP address is an IPv4 or IPv6 address that is shared among a group of routers that run HSRP.

When you configure HSRP on a network segment, you provide a virtual MAC address and a virtual IP address for the HSRP group. You configure the same virtual address on each HSRP-enabled interface in the group. You also configure a unique IP address and MAC address on each interface that acts as the real address. HSRP selects one of these interfaces to be the active router. The active router receives and routes packets destined for the virtual MAC address of the group.

HSRP detects when the designated active router fails. At that point, a selected standby router assumes control of the virtual MAC and IP addresses of the HSRP group. HSRP also selects a new standby router at that time.

HSRP uses a priority designator to determine which HSRP-configured interface becomes the default active router. To configure an interface as the active router, you assign it with a priority that is higher than the priority of all the other HSRP-configured interfaces in the group. The default priority is 100, so if you configure just one interface with a higher priority, that interface becomes the default active router.

Interfaces that run HSRP send and receive multicast User Datagram Protocol (UDP)-based hello messages to detect a failure and to designate active and standby routers. When the active router fails to send a hello message within a configurable period of time, the standby router with the highest priority becomes the active router. The transition of packet forwarding functions between the active and standby router is completely transparent to all hosts on the network.

You can configure multiple HSRP groups on an interface. The virtual router does not physically exist but represents the common default router for interfaces that are configured to provide backup to each other. You do not need to configure the hosts on the LAN with the IP address of the active router. Instead, you configure them with the IP address of the virtual router (virtual IP address) as their default router. If the active router fails to send a hello message within the configurable period of time, the standby router takes over, responds to the virtual addresses, and becomes the active router, assuming the active router duties. From the host perspective, the virtual router remains the same.



Note

Packets received on a routed port destined for the HSRP virtual IP address terminate on the local router, regardless of whether that router is the active HSRP router or the standby HSRP router. This process includes ping and Telnet traffic. Packets received on a Layer 2 (VLAN) interface destined for the HSRP virtual IP address terminate on the active router.

Guidelines and Limitations

Follow these guidelines and limitations:

- The HSRP state must be the same for both HSRP IPv4 and IPv6. The priority and preemption must be configured to result in the same state after failovers.
- Currently, only one IPv4 and one IPv6 group is supported on the same sub-interface in Cisco ACI.
- BFD IPv4 and IPv6 is supported
- Users must configure the same MAC address for IPv4 and IPv6 HSRP groups for dual stack configurations.
- HSRP VIP must be in the same subnet as the interface IP.
- It is recommended that you configure interface delay for HSRP configurations.
- HSRP is only supported on routed-interface or sub-interface. HSRP is not supported on switched virtual interface (SVI).
- Object tracking on HSRP is not supported.
- HSRP is not supported on SVI, therefore no VPC support for HSRP is available.
- HSRP Management Information Base (MIB) for SNMP is not supported.
- Multiple group optimization (MGO) is not supported with HSRP.
- ICMP IPv4 and IPv6 redirects are not supported.
- High availability and Non-Stop Forwarding (NSF) are not supported because HSRP is non restartable in the Cisco ACI environment.
- There is no extended hold-down timer support as HSRP is supported only on leaf switches. HSRP is not supported on spine switches.
- HSRP version change is not supported in APIC. You must remove the configuration and reconfigure.
- HSRP version 2 does not interoperate with HSRP version 1. An interface cannot operate both version 1 and version 2 because both versions are mutually exclusive. However, the different versions can be run on different physical interfaces of the same router.

Configuring HSRP in APIC Using REST API

HSRP is enabled when the leaf switch is configured.

Before You Begin

- The tenant and VRF must be configured.
- VLAN pools must be configured with the appropriate VLAN range defined and the appropriate Layer 3 domain created and attached to the VLAN pool.
- The Attach Entity Profile must also be associated with the Layer 3 domain.
- The interface profile for the leaf switches must be configured as required.

Procedure

- Step 1** Create port selectors.

Example:

```
<polUni>
  <infraInfra dn="uni/infra">
    <infraNodeP name="TenantNode_101">
      <infraLeafS name="leafselector" type="range">
        <infraNodeBlk name="nodeblk" from_="101" to_="101">
          </infraNodeBlk>
        </infraLeafS>
        <infraRsAccPortP tDn="uni/infra/accportprof-TenantPorts_101"/>
      </infraNodeP>
      <infraAccPortP name="TenantPorts_101">
        <infraHPortS name="portselector" type="range">
          <infraPortBlk name="portblk" fromCard="1" toCard="1" fromPort="41" toPort="41">
            </infraPortBlk>
          <infraRsAccBaseGrp tDn="uni/infra/funcprof/accportgrp-TenantPortGrp_101"/>
        </infraHPortS>
      </infraAccPortP>
      <infraFuncP>
        <infraAccPortGrp name="TenantPortGrp_101">
          <infraRsAttEntP tDn="uni/infra/attentp-AttEntityProfTenant"/>
          <infraRsHIfPol tnFabricHIfPolName="default"/>
        </infraAccPortGrp>
      </infraFuncP>
    </infraInfra>
  </polUni>
```

- Step 2** Create a tenant policy.

Example:

```
<polUni>
  <fvTenant name="t9" dn="uni/tn-t9" descr="">
    <fvCtx name="t9_ctx1" pcEnfPref="unenforced">
      </fvCtx>
    <fvBD name="t9_bd1" unkMacUcastAct="flood" arpFlood="yes">
      <fvRsCtx tnFvCtxName="t9_ctx1"/>
      <fvSubnet ip="101.9.1.1/24" scope="shared"/>
    </fvBD>
    <l3extOut dn="uni/tn-t9/out-l3extOut1" enforceRtctrl="export" name="l3extOut1">
      <l3extLNodeP name="Node101">
        <l3extRsNodeL3OutAtt rtrId="210.210.121.121" rtrIdLoopBack="no">
          tDn="topology/pod-1/node-101"/>
        </l3extLNodeP>
        <l3extRsEctx tnFvCtxName="t9_ctx1"/>
        <l3extRsL3DomAtt tDn="uni/l3dom-dom1"/>
        <l3extInstP matchT="AtleastOne" name="extEpg" prio="unspecified" targetDscp="unspecified">
          <l3extSubnet aggregate="" descr="" ip="176.21.21.21/21" name="" scope="import-security"/>
        </l3extInstP>
      </l3extOut>
    </fvTenant>
  </polUni>
```

- Step 3** Create an HSRP interface policy.

Example:

```
<polUni>
  <fvTenant name="t9" dn="uni/tn-t9" descr="">
    <hsrpIfPol name="hsrpIfPol" ctrl="bfd" delay="4" reloadDelay="11"/>
```

```
</fvTenant>
</polUni>
```

Step 4 Create an HSRP group policy.

Example:

```
<polUni>
  <fvTenant name="t9" dn="uni/tn-t9" descr="">
    <hsrpIfPol name="hsrpIfPol" ctrl="bfd" delay="4" reloadDelay="11"/>
  </fvTenant>
</polUni>
```

Step 5 Create an HSRP interface profile and an HSRP group profile.

Example:

```
<polUni>
  <fvTenant name="t9" dn="uni/tn-t9" descr="">
    <l3extOut dn="uni/tn-t9/out-l3extOut1" enforceRtctrl="export" name="l3extOut1">
      <l3extLNodeP name="Node101">
        <l3extLIfP name="eth1-41-v6" ownerKey="" ownerTag="" tag="yellow-green">
          <hsrpIfP name="eth1-41-v6" version="v2">
            <hsrpRsIfPol tnHsrpIfPolName="hsrpIfPol"/>
            <hsrpGroupP descr="" name="HSRPV6-2" groupId="330" groupAf="ipv6" ip="fe80::3"
mac="00:00:0C:18:AC:01" ipObtainMode="admin">
              <hsrpRsGroupPol tnHsrpGroupPolName="G1"/>
            </hsrpGroupP>
          </hsrpIfP>
        </l3extLIfP>
        <l3extRsPathL3OutAtt addr="2002::100/64" descr="" encaps="unknown" encapsScope="local"
ifInstT="l3-port" llAddr="::" mac="00:22:BD:F8:19:FF" mode="regular" mtu="inherit"
tDn="topology/pod-1/paths-101/pathep-[eth1/41]" targetDscp="unspecified">
          <l3extIp addr="2004::100/64"/>
        </l3extRsPathL3OutAtt>
      </l3extLNodeP>
      <l3extLIfP name="eth1-41-v4" ownerKey="" ownerTag="" tag="yellow-green">
        <hsrpIfP name="eth1-41-v4" version="v1">
          <hsrpRsIfPol tnHsrpIfPolName="hsrpIfPol"/>
          <hsrpGroupP descr="" name="HSRPV4-2" groupId="51" groupAf="ipv4" ip="177.21.21.21"
mac="00:00:0C:18:AC:01" ipObtainMode="admin">
            <hsrpRsGroupPol tnHsrpGroupPolName="G1"/>
          </hsrpGroupP>
        </hsrpIfP>
      <l3extRsPathL3OutAtt addr="177.21.21.11/24" descr="" encaps="unknown"
encapsScope="local" ifInstT="l3-port" llAddr="::" mac="00:22:BD:F8:19:FF" mode="regular"
mtu="inherit" tDn="topology/pod-1/paths-101/pathep-[eth1/41]" targetDscp="unspecified">
        <l3extIp addr="177.21.23.11/24"/>
      </l3extRsPathL3OutAtt>
    </l3extLIfP>
  </l3extLNodeP>
</l3extOut>
</fvTenant>
</polUni>
```

IP Multicast

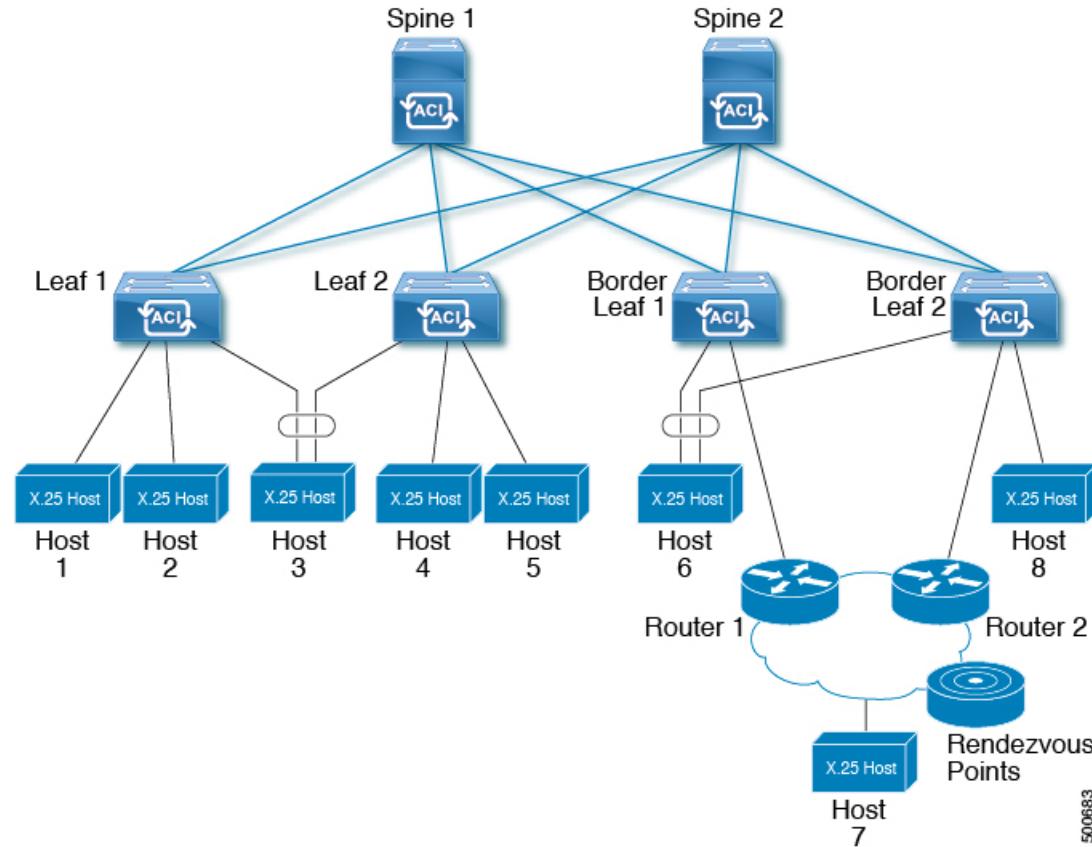
Layer 3 Multicast

All external connectivity to the ACI fabric is maintained through border leaf switches. This holds true for unicast peering and for multicast routing. In most cases, the unicast routing and multicast routing operates together on the same border leaf switches with the multicast protocol operating over the unicast routing protocols.

In this architecture, only the border leaf switches run the full Protocol Independent Multicast (PIM) protocol. Non-border leaf switches run PIM in a passive mode on the interfaces. They do not peer with any other PIM routers. The border leaf switches peer with other PIM routers connected to them over L3 Outs and also with each other.

The following figure shows the border leaf (BL) switches (BL1 and BL2) connecting to routers (R1 and R2) in the multicast cloud. Each virtual routing and forwarding (VRF) in the fabric that requires multicast routing will peer separately with external multicast routers.

Figure 32: Overview of Multicast Cloud



Guidelines for Configuring Layer 3 Multicast

See the following guidelines:

- The Layer 3 multicast configuration is done at the VRF level so protocols function within the VRF and multicast is enabled in a VRF, and each multicast VRF can be turned on or off independently.
- Once a VRF is enabled for multicast, the individual bridge domains (BDs) and L3 Outs under the enabled VRF can be enabled for multicast configuration. By default, multicast is disabled in all BDs and Layer 3 Outs.
- Layer 3 multicast is not currently supported on VRFs that are configured with a shared L3 Out.
- Any Source Multicast (ASM) and Source-Specific Multicast (SSM) are supported.

- Bidirectional PIM, Rendezvous Point (RP) within the ACI fabric, and PIM IPv6 are currently not supported.
- IGMP snooping cannot be disabled on pervasive bridge domains with multicast routing enabled.
- Multicast routers are not supported in pervasive bridge domains.
- The Layer 3 multicast feature is currently supported on the N9K-93180YC-EX model leaf switch.
- Layer 3 Out ports and sub-interfaces are supported while external SVIs are not supported. Since external SVIs are not supported, PIM cannot be enabled in L3-VPC.
- For Layer 3 multicast support for multipod, when the ingress leaf switch receives a packet from a source attached on a bridge domain that is enabled for multicast routing, the ingress leaf switch sends only a routed VRF copy to the fabric (routed implies that the TTL is decremented by 1, and the source-mac is rewritten with a pervasive subnet MAC). The egress leaf switch also routes the packet into receivers in all the relevant bridge domains. Therefore, if a receiver is on the same bridge domain as the source, but on a different leaf switch than the source, that receiver continues to get a routed copy, even though it is in the same bridge domain.

For more information, see details about layer 3 multicast support for multipod that leverages existing Layer 2 design, at the following link [Adding Pods](#).

Configuring Layer 3 Multicast Using REST API

Procedure

Step 1 Configure tenant, VRF, and enable multicast on VRF.

Example:

```
<fvTenant dn="uni/tn-PIM_Tenant" name="PIM_Tenant">
  <fvCtx knwMcastAct="permit" name="ctx1">
    <pimCtxP mtu="1500">
      </pimCtxP>
    </fvCtx>
  </fvTenant>
```

Step 2 Configure L3 Out and enable multicast (PIM, IGMP) on L3 Out.

Example:

```
<l3extOut enforceRtctrl="export" name="l3out-pim_l3out1">
  <l3extRsEctx tnFvCtxName="ctx1"/>
  <l3extLNodeP configIssues="" name="bLeaf-CTX1-101">
    <l3extRsNodeL3OutAtt rtrId="200.0.0.1" rtrIdLoopBack="yes"
    tDn="topology/pod-1/node-101"/>
    <l3extLIfP name="if-PIM_Tenant-CTX1" tag="yellow-green">
      <igmpIfP/>
      <pimIfP>
        <pimRsIfPol tDn="uni/tn-PIM_Tenant/pimifpol-pim_pol1"/>
      </pimIfP>
      <l3extRsPathL3OutAtt addr="131.1.1.1/24" ifInstT="l3-port" mode="regular"
      mtu="1500" tDn="topology/pod-1/paths-101/pathep-[eth1/46]"/>
    </l3extLIfP>
  </l3extLNodeP>
  <l3extRsL3DomAtt tDn="uni/l3dom-l3outDom"/>
  <l3extInstP name="l3out-PIM_Tenant-CTX1-1topo" >
  </l3extInstP>
```

```

<pimExtP enabledAf="ipv4-mcast" name="pim"/>
</l3extOut>
```

Step 3 Configure BD under tenant and enable multicast and IGMP on BD.

Example:

```

<fvTenant dn="uni/tn-PIM_Tenant" name="PIM_Tenant">
    <fvBD arpFlood="yes" mcastAllow="yes" multiDstPktAct="bd-flood" name="bd2" type="regular"
        unicastRoute="yes" unkMacUcastAct="flood" unkMcastAct="flood">
        <igmpIfP/>
        <fvRsBDToOut tnL3extOutName="l3out-pim_l3out1"/>
        <fvRsCtx tnFvCtxName="ctx1"/>
        <fvRsIgmpsN/>
        <fvSubnet ctrl="" ip="41.1.1.254/24" preferred="no" scope="private" virtual="no"/>
    </fvBD>
</fvTenant>
```

Step 4 Configure IGMP policy and assign it to BD.

Example:

```

<fvTenant dn="uni/tn-PIM_Tenant" name="PIM_Tenant">
    <igmpIfPol grpTimeout="260" lastMbrCnt="2" lastMbrRespTime="1" name="igmp_pol"
        querierTimeout="255" queryIntvl="125" robustFac="2" rspIntvl="10" startQueryCnt="2"
        startQueryIntvl="125" ver="v2">
        </igmpIfPol>
        <fvBD arpFlood="yes" mcastAllow="yes" name="bd2">
            <igmpIfP>
                <igmpRsIfPol tDn="uni/tn-PIM_Tenant/igmpIfPol-igmp_pol"/>
            </igmpIfP>
        </fvBD>
    </fvTenant>
```

Step 5 Configure route map, PIM, and RP policy on VRF.

Example:

```

<fvTenant dn="uni/tn-PIM_Tenant" name="PIM_Tenant">
    <pimRouteMapPol name="rootMap">
        <pimRouteMapEntry action="permit" grp="224.0.0.0/4" order="10" rp="0.0.0.0"
            src="0.0.0.0/0"/>
    </pimRouteMapPol>
    <fvCtx knwMcastAct="permit" name="ctx1">
        <pimCtxtP ctrl="" mtu="1500">
            <pimStaticRPPol>
                <pimStaticRPEntryPol rpIp="131.1.1.2">
                    <pimRPGrpRangePol>
                        <rtdmcRsFilterToRtMapPol tDn="uni/tn-PIM_Tenant/romap-rootMap"/>
                    </pimRPGrpRangePol>
                </pimStaticRPEntryPol>
            </pimStaticRPPol>
        </pimCtxtP>
    </fvCtx>
</fvTenant>
```

Step 6 Configure PIM interface policy and apply it on L3 Out.

Example:

```

<fvTenant dn="uni/tn-PIM_Tenant" name="PIM_Tenant">
    <pimIfPol authKey="" authT="none" ctrl="" drDelay="60" drPrio="1" helloItvl="30000"
        itvl="60" name="pim_pol1"/>
    <l3extOut enforceRtctrl="export" name="l3out-pim_l3out1" targetDscp="unspecified">
        <l3extRsEctx tnFvCtxName="ctx1"/>
        <l3extLNodeP name="bLeaf-CTX1-101">
            <l3extRsNodeL3OutAtt rtrId="200.0.0.1" rtrIdLoopBack="yes"
                tDn="topology/pod-1/node-101"/>
                <l3extLifP name="if-SIRI_VPC_src_recv-CTX1" tag="yellow-green">
                    <pimIfP>
```

```

<pimRsIfPol tDn="uni/tn-tn-PIM_Tenant/pimifpol-pim_pol1"/>
</pimIfP>
</l3extLIfP>
</l3extLNodeP>
</l3extOut>
</fvTenant>

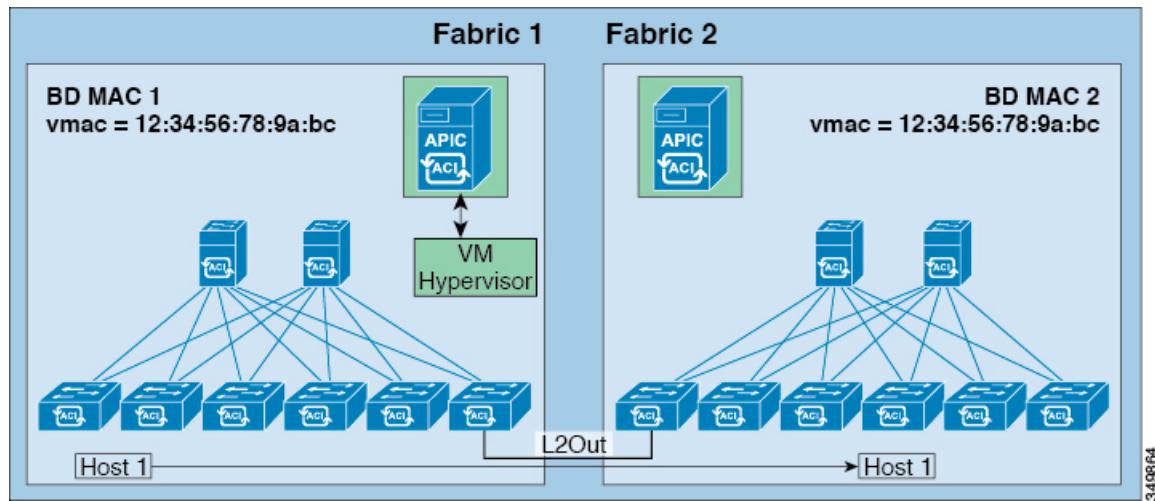
```

Pervasive Gateway

Common Pervasive Gateway

Multiple ACI fabrics can be configured with an IPv4 common gateway on a per bridge domain basis. Doing so enables moving one or more virtual machines (VM) or conventional hosts across the fabrics while the host retains its IP address. VM host moves across fabrics can be done automatically by the VM hypervisor. The ACI fabrics can be co-located, or provisioned across multiple sites. The Layer 2 connection between the ACI fabrics can be a local link, or can be across a routed WAN link. The following figure illustrates the basic common pervasive gateway topology.

Figure 33: ACI Multi-Fabric Common Pervasive Gateway



The per-bridge domain common pervasive gateway configuration requirements are as follows:

- The bridge domain MAC (*mac*) values for each fabric must be unique.



Note

The default bridge domain MAC (*mac*) address values are the same for all ACI fabrics. The common pervasive gateway requires an administrator to configure the bridge domain MAC (*mac*) values to be unique for each ACI fabric.

- The bridge domain virtual MAC (*vmac*) address and the subnet virtual IP address must be the same across all ACI fabrics for that bridge domain. Multiple bridge domains can be configured to communicate

across connected ACI fabrics. The virtual MAC address and the virtual IP address can be shared across bridge domains.

Configuring Common Pervasive Gateway Using the REST API

Before You Begin

- The tenant, VRF, and bridge domain are created.

Procedure

Configure Common Pervasive Gateway.

Example:

```
<!--Things that are bolded only matters-->
<?xml version="1.0" encoding="UTF-8"?>
<!-- api/policymgr/mo/.xml -->
<polUni>
  <fvTenant name="test">
    <fvCtx name="test"/>

    <fvBD name="test" vmac="12:34:56:78:9a:bc"

```

Explicit Prefix Lists

About Explicit Prefix List Support for Route Maps/Profile

In Cisco APIC, for public bridge domain (BD) subnets and external transit networks, inbound and outbound route controls are provided through an explicit prefix list. Inbound and outbound route control for Layer 3 Out is managed by the route map/profile (rtctrlProfile). The route map/profile policy supports a fully controllable prefix list for Layer 3 Out in the Cisco ACI fabric.

The subnets in the prefix list can represent the bridge domain public subnets or external networks. Explicit prefix list presents an alternate method and can be used instead of the following:

- Advertising BD subnets through BD to Layer 3 Out relation.

**Note**

The subnet in the BD must be marked public for the subnet to be advertised out.

- Specifying a subnet in the l3extInstP with export/import route control for advertising transit and external networks.

**Note**

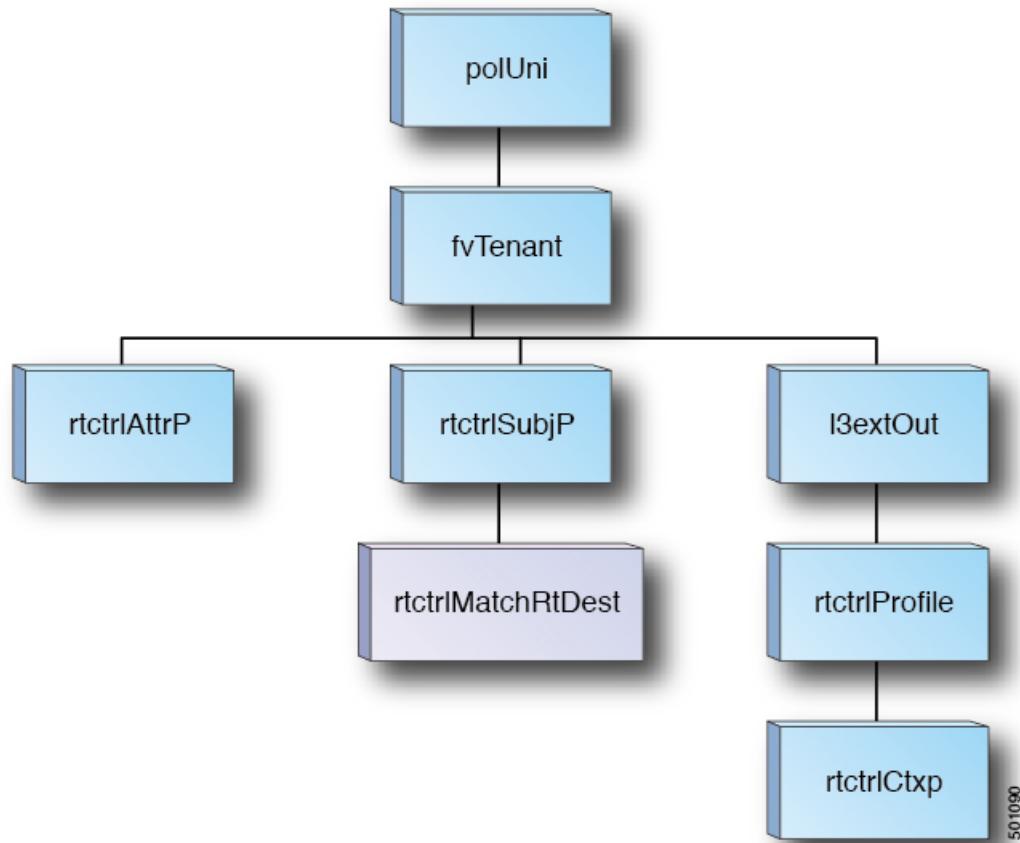
The explicit prefix list feature is supported for import and export route control only for BGP in the current Cisco APIC software release 2.1(x).

Explicit prefix list is defined through a new match type that is called match route destination (rtctrlMatchRtDest). An example usage is provided in the API example that follows.

**Note**

For detailed information about route maps, route import and export, route summarization, and route community match, see the *Cisco Application Centric Infrastructure Fundamentals*.

Figure 34: External Policy Model of API



501090

Additional information about match rules, set rules when using explicit prefix list are as follows:

Match Rules

- Under the tenant (fvTenant), you can create match profiles (rtctrlSubjP) for route map filtering. Each match profile can contain one or more match rules. Match rule supports multiple match types. Prior to Cisco APIC release 2.1(x), match types supported were explicit prefix list and community list.

Starting with Cisco APIC release 2.1(x), explicit prefix match or match route destination (rtctrlMatchRtDest) is supported.

Match prefix list (rtctrlMatchRtDest) supports one or more subnets with an optional aggregate flag. Aggregate flags are used for allowing prefix matches with multiple masks starting with the mask mentioned in the configuration till the maximum mask allowed for the address family of the prefix . This is the equivalent of the "le " option in the prefix-list in NX-OS software (example, 10.0.0.0/8 le 32).

The prefix list can be used for covering the following cases:

- Allow all (0.0.0.0/0 with aggregate flag. equivalent of "0.0.0.0/0 le 32")
- One or more of specific prefixes (example: 10.1.1.0/24)
- One or more of prefixes with aggregate flag (example, equivalent of 10.1.1.0/24 le 32).
- The explicit prefix match rules can contain one or more subnets, and these subnets can be bridge domain public subnets or external networks. Subnets can also be aggregated up to the maximum subnet mask (/32 for IPv4 and /128 for IPv6).
- When multiple match rules of different types are present (such as match community and explicit prefix match), the match rule is allowed only when the match statements of all individual match types match. This is the equivalent of the AND filter. The explicit prefix match is contained by the subject profile (rtctrlSubjP) and will form a logical AND if other match rules are present under the subject profile.
- Within a given match type (such as match prefix list), at least one of the match rules statement must match. Multiple explicit prefix match (rtctrlMatchRtDest) can be defined under the same subject profile (rtctrlSubjP) which will form a logical OR.

Set Rules

- Set policies must be created to define set rules that are carried with the explicit prefixes such as set community, set tag.

Guidelines and Limitations

- You must choose one of the following two methods to configure your route maps. If you use both methods, it will result in double entries and undefined route maps.
 - Add routes under the bridge domain (BD) and configure a BD to Layer 3 Outside relation
 - Configure the match prefix under rtctrlSubjP match profiles.

About Route Map/Profile

The route profile is a logical policy that defines an ordered set (rtctrlCtxP) of logical match action rules with associated set action rules. The route profile is the logical abstract of a route map. Multiple route profiles can be merged into a single route map. A route profile can be one of the following types:

- Match Prefix and Routing Policy: Pervasive subnets (fvSubnet) and external subnets (l3extSubnet) are combined with a route profile and merged into a single route map (or route map entry). Match Prefix and Routing Policy is the default value.
- Match Routing Policy Only: The route profile is the only source of information to generate a route map, and it will overwrite other policy attributes.



Note

- When explicit prefix list is used, the type of the route profile should be set to "match routing policy only".
- Explicit prefix list is currently supported only for BGP Layer 3 Out route-control.

After the match and set profiles are defined, the route map must be created in the Layer 3 Out. Route maps can be created using one of the following methods:

- Create a "default-export" route map for export route control, and a "default-import" route map for import route control.
- Create other route maps (not named default-export or default-import) and setup the relation from one or more l3extInstPs or subnets under the l3extInstP.
- In either case, match the route map on explicit prefix list by pointing to the rtctrlSubjP within the route map.

In the export and import route map, the set and match rules are grouped together along with the relative sequence across the groups (rtctrlCtxP). Additionally, under each group of match and set statements (rtctrlCtxP) the relation to one or more match profiles are available (rtctrlSubjP).

Any protocol enabled on Layer 3 Out (for example BGP protocol), will use the export and import route map for route filtering.

Aggregation Support for Explicit Prefix List

Each prefix (rtctrlMatchRtDest) in the match prefixes list can be aggregated to support multiple subnets matching with one prefix list entry..

Aggregated prefixes and BD private subnets: Although subnets in the explicit prefix list match may match the BD private subnets using aggregated or exact match, private subnets will not be advertised through the routing protocol using the explicit prefix list. The scope of the BD subnet must be set to "public" for the explicit prefix list feature to advertise the BD subnets.

Configuring Route Map/Profile with Explicit Prefix List Using REST API

Before You Begin

- Tenant and VRF must be configured.

Procedure

Configure the route map/profile using explicit prefix list.

Example:

```

<fvTenant name="PM" status="">
    <rtctrlAttrP name="set_dest">
        <rtctrlSetComm community="regular:as2-nn2:5:24"/>
    </rtctrlAttrP>

    <rtctrlSubjP name="match_dest">
        <rtctrlMatchRtDest ip="192.169.0.0/24"/>

        <rtctrlMatchCommTerm name="term1">
            <rtctrlMatchCommFactor community="regular:as2-nn2:5:24" status="" />
            <rtctrlMatchCommFactor community="regular:as2-nn2:5:25" status="" />
        </rtctrlMatchCommTerm>

        <rtctrlMatchCommRegexTerm commType="regular" regex="200:.*" status="" />
    </rtctrlSubjP>

    <fvCtx name="ctx">
    </fvCtx>

<l3extOut name="L3Out_1" enforceRtctrl="import,export" status="">

    <l3extRsEctx tnFvCtxName="ctx" />
    <l3extLNodeP name="bLeaf">
        <l3extRsNodeL3OutAtt tDn="topology/pod-1/node-101" rtrId="1.2.3.4"/>
        <l3extLIfP name="portIf">
            <l3extRsPathL3OutAtt tDn="topology/pod-1/paths-101/pathep-[eth1/25]" ifInstT="sub-interface" encap="vlan-1503" addr="10.11.12.11/24" />
            <ospfIfP/>
            </l3extLIfP>
        <bgpPeerP addr="5.16.57.18/32" ctrl="send-com"/>
        <bgpPeerP addr="6.16.57.18/32" ctrl="send-com"/>
    </l3extLNodeP>

    <bgpExtP />
    <ospfExtP areaId="0.0.0.59" areaType="nssa" status="" />

    <l3extInstP name="l3extInstP_1" status="">
        <l3extSubnet ip="17.11.1.11/24" scope="import-security" />

    </l3extInstP>

    <rtctrlProfile name="default-export" type="global" status="">
        <rtctrlCtxP name="ctx1" order="2">
            <rtctrlRsCtxPToSubjP tnRtctrlSubjPName="match_dest" status="" />
            <rtctrlScope name="scope" status="">
                <rtctrlRsScopeToAttrP tnRtctrlAttrPName="set_dest" status="" />
            </rtctrlScope>
        </rtctrlCtxP>
    </rtctrlProfile>
</fvTenant>

```

```

        </rtctrlProfile>

    </l3extOut>
    <fvBD name="testBD">
        <fvRsBDToOut tnL3extOutName="L3Out_1" />
        <fvRsCtx tnFvCtxName="ctx" />
        <fvSubnet ip="40.1.1.12/24" scope="public" />
        <fvSubnet ip="40.1.1.2/24" scope="private" />
        <fvSubnet ip="2003::4/64" scope="public" />
    </fvBD>
</fvTenant>

```

IP Address Aging Tracking

Overview

The IP aging policy tracks and ages unused IPs on an endpoint. Tracking is performed using the endpoint retention policy configured for the BD to send ARP requests (for IPv4) and neighbor solicitations (for IPv6) at 75% of the local endpoint aging interval. When no response is received from an IP, that IP is aged out.

This document explains how to configure the IP aging policy.

Configuring IP Aging Using the REST API

This section explains how to enable and disable the IP aging policy using the REST API.

Procedure

Step 1 To enable the IP aging policy:

Example:

```
<epIpAgingP adminSt="enabled" descr="" dn="uni/infra/ipAgingP-default" name="default"
ownerKey="" ownerTag="" />
```

Step 2 To disable the IP aging policy:

Example:

```
<epIpAgingP adminSt="disabled" descr="" dn="uni/infra/ipAgingP-default" name="default"
ownerKey="" ownerTag="" />
```

Route Summarization

Configuring Route Summarization for BGP, OSPF, and EIGRP Using the REST API

Procedure

Step 1 Configure BGP route summarization using the REST API as follows:

Example:

```

<fvTenant name="common">
    <fvCtx name="vrf1"/>
    <bgpRtSummPol name="bgp_rt_summ" cntrl='as-set' />
    <l3extOut name="l3_ext_pol" >
        <l3extLNodeP name="bLeaf">
            <l3extRsNodeL3OutAtt tDn="topology/pod-1/node-101" rtrId="20.10.1.1"/>
            <l3extLIfP name='portIf'>
                <l3extRsPathL3OutAtt tDn="topology/pod-1/paths-101/pathep-[eth1/31]" ifInstT='l3-port' addr="10.20.1.3/24"/>
            </l3extLIfP>
        </l3extLNodeP>
    <bgpExtP />
        <l3extInstP name="InstP" >
            <l3extSubnet ip="10.0.0.0/8" scope="export-rtctrl">
                <l3extRsSubnetToRtSumm tDn="uni/tn-common/bgpRtsumm-bgp_rt_summ"/>
                <l3extRsSubnetToProfile tnRtctrlProfileName="rtpprof"/>
            </l3extSubnet>
        </l3extInstP>
        <l3extRsEctx tnFvCtxName="vrf1"/>
    </l3extOut>
</fvTenant>

```

Step 2 Configure OSPF inter-area and external summarization using the following REST API:**Example:**

```

<?xml version="1.0" encoding="utf-8"?>
<fvTenant name="t20">
    <!--Ospf Inter External route summarization Policy-->
    <ospfRtSummPol cost="unspecified" interAreaEnabled="no" name="ospfext"/>
    <!--Ospf Inter Area route summarization Policy-->
    <ospfRtSummPol cost="16777215" interAreaEnabled="yes" name="interArea"/>
    <fvCtx name="ctx0" pcEnfDir="ingress" pcEnfPref="enforced"/>
    <!-- L3OUT backbone Area-->
    <l3extOut enforceRtctrl="export" name="l3_1" ownerKey="" ownerTag="" targetDscp="unspecified">
        <l3extRsEctx tnFvCtxName="ctx0"/>
        <l3extLNodeP name="node-101">
            <l3extRsNodeL3OutAtt rtrId="20.1.3.2" rtrIdLoopBack="no" tDn="topology/pod-1/node-101"/>
            <l3extLIfP name="intf-1">
                <l3extRsPathL3OutAtt addr="20.1.5.2/24" encaps="vlan-1001" ifInstT="sub-interface" tDn="topology/pod-1/paths-101/pathep-[eth1/33]"/>
            </l3extLIfP>
        </l3extLNodeP>
        <l3extInstP name="l3InstP1">
            <fvRsProv tnVzBrCPName="default"/>
            <!--Ospf External Area route summarization-->
            <l3extSubnet aggregate="" ip="193.0.0.0/8" name="" scope="export-rtctrl">
                <l3extRsSubnetToRtSumm tDn="uni/tn-t20/ospfrtsumm-ospfext"/>
            </l3extSubnet>
        </l3extInstP>
        <ospfExtP areaCost="1" areaCtrl="redistribute,summary" areaId="backbone" areaType="regular"/>
    </l3extOut>
    <!-- L3OUT Regular Area-->
    <l3extOut enforceRtctrl="export" name="l3_2">
        <l3extRsEctx tnFvCtxName="ctx0"/>
        <l3extLNodeP name="node-101">
            <l3extRsNodeL3OutAtt rtrId="20.1.3.2" rtrIdLoopBack="no" tDn="topology/pod-1/node-101"/>
            <l3extLIfP name="intf-2">
                <l3extRsPathL3OutAtt addr="20.1.2.2/24" encaps="vlan-1014" ifInstT="sub-interface" tDn="topology/pod-1/paths-101/pathep-[eth1/11]"/>
            </l3extLIfP>
        </l3extLNodeP>
    </l3extOut>

```

```

</l3extLNodeP>
<l3extInstP matchT="AtleastOne" name="l3InstP2">
    <fvRsCons tnVzBrCPName="default"/>
    <!--Ospf Inter Area route summarization-->
    <l3extSubnet aggregate="" ip="197.0.0.0/8" name="" scope="export-rtctrl">
        <l3extRsSubnetToRtSumm tDn="uni/tn-t20/ospfrtsumm-interArea"/>
    </l3extSubnet>
</l3extInstP>
<ospfExtB areaCost="1" areaCtrl="redistribute,summary" areaId="0.0.0.57" areaType="regular"/>
</l3extOut>
</fvTenant>

```

Step 3 Configure EIGRP summarization using the following REST API:

Example:

```

<fvTenant name="exampleCorp">
    <l3extOut name="out1">
        <l3extInstP name="eigrpSummInstp" >
            <l3extSubnet aggregate="" descr="" ip="197.0.0.0/8" name="" scope="export-rtctrl">
                <l3extRsSubnetToRtSumm/>
            </l3extSubnet>
        </l3extInstP>
    </l3extOut>
    <eigrpRtSummPol name="pol1" />

```

Note There is no route summarization policy to be configured for EIGRP. The only configuration needed for enabling EIGRP summarization is the summary subnet under the InstP.

Configuring Route Summarization for BGP, OSPF, and EIGRP Using the REST API

Procedure

Step 1 Configure BGP route summarization using the REST API as follows:

Example:

```

<fvTenant name="common">
    <fvCtx name="vrf1"/>
    <bgpRtSummPol name="bgp_rt_summ" cntrl='as-set' />
    <l3extOut name="l3_ext_pol" >
        <l3extLNodeP name="bLeaf">
            <l3extRsNodeL3OutAtt tDn="topology/pod-1/node-101" rtrId="20.10.1.1"/>
            <l3extLifP name='portIf'>
                <l3extRsPathL3OutAtt tDn="topology/pod-1/paths-101/pathep-[eth1/31]" ifInstT='l3-port' addr="10.20.1.3/24"/>
            </l3extLifP>
        </l3extLNodeP>
        <bgpExtP />
        <l3extInstP name="InstP" >
            <l3extSubnet ip="10.0.0.0/8" scope="export-rtctrl">
                <l3extRsSubnetToRtSumm tDn="uni/tn-common/bgpRtsum-bgp_rt_summ"/>
                <l3extRsSubnetToProfile tnRtctrlProfileName="rtprof"/>
            </l3extSubnet>
        </l3extInstP>
        <l3extRsEctx tnFvCtxName="vrf1"/>
    </l3extOut>

```

```
</fvTenant>
```

- Step 2** Configure OSPF inter-area and external summarization using the following REST API:

Example:

```
<?xml version="1.0" encoding="utf-8"?>
<fvTenant name="t20">
    <!--Ospf Inter External route summarization Policy-->
    <ospfRtSummPol cost="unspecified" interAreaEnabled="no" name="ospfext"/>
    <!--Ospf Inter Area route summarization Policy-->
    <ospfRtSummPol cost="16777215" interAreaEnabled="yes" name="interArea"/>
    <fvCtx name="ctx0" pcEnfDir="ingress" pcEnfPref="enforced"/>
    <!-- L3OUT backbone Area-->
    <l3extOut enforceRtctrl="export" name="l3_1" ownerKey="" ownerTag="" targetDscp="unspecified">
        <l3extRsEctx tnFvCtxName="ctx0"/>
        <l3extLNodeP name="node-101">
            <l3extRsNodeL3OutAtt rtrId="20.1.3.2" rtrIdLoopBack="no" tDn="topology/pod-1/node-101"/>

            <l3extLIfP name="intf-1">
                <l3extRsPathL3OutAtt addr="20.1.5.2/24" encaps="vlan-1001" ifInstT="sub-interface" tDn="topology/pod-1/paths-101/pathep-[eth1/33]"/>
            </l3extLIfP>
        </l3extLNodeP>
        <l3extInstP name="l3InstP1">
            <fvRsProv tnVzBrCPName="default"/>
            <!--Ospf External Area route summarization-->
            <l3extSubnet aggregate="" ip="193.0.0.0/8" name="" scope="export-rtctrl">
                <l3extRsSubnetToRtSumm tDn="uni/tn-t20/ospfrtsumm-ospfext"/>
            </l3extSubnet>
        </l3extInstP>
        <ospfExtP areaCost="1" areaCtrl="redistribute,summary" areaId="backbone" areaType="regular"/>
    </l3extOut>
    <!-- L3OUT Regular Area-->
    <l3extOut enforceRtctrl="export" name="l3_2">
        <l3extRsEctx tnFvCtxName="ctx0"/>
        <l3extLNodeP name="node-101">
            <l3extRsNodeL3OutAtt rtrId="20.1.3.2" rtrIdLoopBack="no" tDn="topology/pod-1/node-101"/>

            <l3extLIfP name="intf-2">
                <l3extRsPathL3OutAtt addr="20.1.2.2/24" encaps="vlan-1014" ifInstT="sub-interface" tDn="topology/pod-1/paths-101/pathep-[eth1/11]"/>
            </l3extLIfP>
        </l3extLNodeP>
        <l3extInstP matchT="AtleastOne" name="l3InstP2">
            <fvRsCons tnVzBrCPName="default"/>
            <!--Ospf Inter Area route summarization-->
            <l3extSubnet aggregate="" ip="197.0.0.0/8" name="" scope="export-rtctrl">
                <l3extRsSubnetToRtSumm tDn="uni/tn-t20/ospfrtsumm-interArea"/>
            </l3extSubnet>
        </l3extInstP>
        <ospfExtP areaCost="1" areaCtrl="redistribute,summary" areaId="0.0.0.57" areaType="regular"/>
    </l3extOut>
</fvTenant>
```

- Step 3** Configure EIGRP summarization using the following REST API:

Example:

```
<fvTenant name="exampleCorp">
    <l3extOut name="out1">
        <l3extInstP name="eigrpSummInstp" >
            <l3extSubnet aggregate="" descr="" ip="197.0.0.0/8" name="" scope="export-rtctrl">
                <l3extRsSubnetToRtSumm/>
            </l3extSubnet>
        </l3extInstP>
    </l3extOut>
</fvTenant>
```

```

</l3extInstP>
</l3extOut>
<eigrpRtSummPol name="pol1" />

```

Note There is no route summarization policy to be configured for EIGRP. The only configuration needed for enabling EIGRP summarization is the summary subnet under the InstP.

External Route Interleak

Overview

This topic provides a typical example of how to configure an interleak of external routes such as OSPF or EIGRP into BGP when using Cisco APIC.

Interleak from EIGRP or OSPF has been available in earlier releases. The feature now enables the user to set attributes, such as community, preference, and metric for route leaking from EIGRP or OSPF to BGP.

Configuring Interleak of External Routes Using the REST API

Before You Begin

- The tenant, VRF, and bridge domain are created.
- The external routed domain is created.

Procedure

Configure an interleak of external routes:

Example:

```

<l3extOut descr="" enforceRtctrl="export" name="out1" ownerKey="" ownerTag=""
targetDscp="unspecified">
    <l3extLNodeP configIssues="" descr="" name="Lnodep1" ownerKey="" ownerTag=""
tag="yellow-green" targetDscp="unspecified">
        <l3extRsNodeL3OutAtt rtrId="1.2.3.4" rtrIdLoopBack="yes"
tDn="topology/pod-1/node-101"/>
        <l3extLIfP descr="" name="lifp1" ownerKey="" ownerTag="" tag="yellow-green">
            <ospfIfP authKeyId="1" authType="none" descr="" name="">
                <ospfRsIfPol tnOspfIfPolName="" />
            </ospfIfP>
            <l3extRsNdIfPol tnNdIfPolName="" />
            <l3extRsIngressQosDppPol tnQosDppPolName="" />
            <l3extRsEgressQosDppPol tnQosDppPolName="" />
            <l3extRsPathL3OutAtt addr="12.12.7.16/24" descr="" encaps="unknown"
encapScope="local" ifInstT="13-port" llAddr="::" mac="00:22:BD:F8:19:FF" mode="regular"
mtu="inherit" tDn="topology/pod-1/paths-101/pathep-[eth1/11]" targetDscp="unspecified"/>
        </l3extLIfP>
    </l3extLNodeP>
    <l3extRsEctx tnFvCtxName="ctx1" />
    <l3extRsInterleakPol tnRtctrlProfileName="interleak" />
    <l3extRsL3DomAtt tDn="uni/l3dom-Domain" />
    <l3extInstP descr="" matchT="AtleastOne" name="InstP1" prio="unspecified"
targetDscp="unspecified">
        <fvRsCustQosPol tnQosCustomPolName="" />
        <l3extSubnet aggregate="" descr="" ip="14.15.16.0/24" name="" />
    </l3extInstP>
</l3extOut>

```

```

scope="export-rtctrl,import-security"/>
</l3extInstP>
<ospfExtP areaCost="1" areaCtrl="redistribute,summary" areaId="0.0.0.1" areaType="nssa">
  descr=""
</l3extOut>

```

Routing Protocols

BGP and BFD

Guidelines for Configuring a BGP Layer 3 Outside Network Connection

When configuring a BGP external routed network, follow these guidelines:

- Whenever a router ID is created on a leaf switch, it creates an internal loopback address. When setting up a BGP connection on a leaf switch, your router ID cannot be the same as the interface IP address as it is not supported on the ACI leaf switch. The router ID must be a different address in a different subnet. On the external Layer 3 device, the router ID can be the loopback address or an interface address. Ensure that the route to leaf router ID is present in the routing table of the the Layer3 device either through static route or OSPF configuration. Also, when setting up the BGP neighbor on a Layer 3 device, the peer IP address that is used must be the router ID of the leaf switch.
- While configuring two external Layer 3 networks with BGP on the same node, loopback addresses must be explicitly defined. Failing to follow this guideline can prevent BGP from being established.
- By definition, the router ID is a loopback interface. To change the router ID and assign a different address for loopback, you must create a loopback interface policy. (The loopback policy can be configured as one for each address family, IPv4 and IPv6.) If you do not wish to create a loopback policy, then you can enable a router ID loopback which is enabled by default. If the router ID loopback is disabled, no loopback is created for the specific Layer 3 outside on which it is deployed.
- This configuration task is applicable for iBGP and eBGP. If the BGP configuration is on a loopback address then it can be an iBGP session or a multi-hop eBGP session. If the peer IP address is for a physical interface where the BGP peer is defined, then the physical interface is used.
- The user must configure an IPv6 address to enable peering over loopback using IPv6.
- The autonomous system feature can only be used for eBGP peers. It enables a router to appear to be a member of a second autonomous system (AS), in addition to its real AS. Local AS allows two ISPs to merge without modifying peering arrangements. Routers in the merged ISP become members of the new autonomous system but continue to use their old AS numbers for their customers.
- Starting with release 1.2(1x), tenant networking protocol policies for BGP l3extOut connections can be configured with a maximum prefix limit that enables monitoring and restricting the number of route prefixes received from a peer. Once the max prefix limit is exceeded, a log entry can be recorded, further prefixes can be rejected, the connection can be restarted if the count drops below the threshold in a fixed interval, or the connection is shut down. Only one option can be used at a time. The default setting is a limit of 20,000 prefixes, after which new prefixes are rejected. When the reject option is deployed, BGP accepts one more prefix beyond the configured limit and the APIC raises a fault.
- When BGP and OSPF are configured for the same Layer 3 Outside, BGP is treated as the primary routing protocol, and OSPF becomes the secondary routing protocol. Therefore, when import route control enforcement is enabled, the subnet with import route control selected under the L3 Outside EPG applies only to the BGP neighbor. Therefore OSPF will allow every subnet advertised by the neighbor. However

if the route has the same route-tag (in comparison with the route-tag policy) of the VRF, it will not be installed in the routing table even if the source is from BGP or OSPF.

- Configuring EIGRP and BGP for the same Layer 3 outside is not supported.
- The number of BGP communities and extended communities must be restricted to a maximum of 32 in the route map. The same restriction applies to community lists and extended community lists in addition to route maps.

BGP Connection Types and Loopback Guidelines

For BGP connection types and loopback set up requirements, follow these guidelines:

- When a router ID is created for a node, a loopback interface with the same IP address as the router ID is also created. This is the default behavior but can be overridden when configuring the router ID.
- The IP address configured for the router ID should be a different address in a different subnet from any other IP address configured on the node.
- The loopback interface with the router ID IP address can be used for peering with an external router if there is only one external BGP peer per node. When peering with multiple BGP peers on the same node, the router ID loopback address must not be used. An explicit loopback interface policy per BGP peer must be used.
- A loopback interface policy is not required when peering with an external router on a directly connected network.
- When peering to an external router with a loopback interface (iBGP or eBGP multi-hop) a static route or OSPF route is required to reach the remote peer loopback address.
- For BGP, the loopback creation is selected by default. When it is selected, the loopback is used as the source interface to establish BGP sessions. However, to establish eBGP over a physical interface, the administrator must not create loopback.

Table 11:

BGP Connection Type	Loopback required	Loopback same as Router ID	Static/OSPF route required
iBGP direct	No	Not applicable	No
iBGP loopback peering	Yes, a separate loopback per BGP peer	No, if multiple Layer 3 out are on the same node	Yes
eBGP direct	No	Not applicable	No
eBGP loopback peering (multi-hop)	Yes, a separate loopback per BGP peer	No, if multiple Layer 3 out are on the same node	Yes

Configuring an MP-BGP Route Reflector Using the REST API

Procedure

- Step 1** Mark the spine switches as route reflectors.

Example:

```
POST https://apic-ip-address/api/policymgr/mo/uni/fabric.xml
```

```
<bgpInstPol name="default">
  <bgpAsP asn="1" />
  <bgpRRP>
    <bgpRRNodePEp id="<spine_id1>" />
    <bgpRRNodePEp id="<spine_id2>" />
  </bgpRRP>
</bgpInstPol>
```

- Step 2** Set up the pod selector using the following post.

Example:

For the FuncP setup—

```
POST https://apic-ip-address/api/policymgr/mo/uni.xml
```

```
<fabricFuncP>
  <fabricPodPGrp name="bgpRRPodGrp">
    <fabricRsPodPGrpBGFRRP tnBgpInstPolName="default" />
  </fabricPodPGrp>
</fabricFuncP>
```

Example:

For the PodP setup—

```
POST https://apic-ip-address/api/policymgr/mo/uni.xml
```

```
<fabricPodP name="default">
  <fabricPodS name="default" type="ALL">
    <fabricRsPodPGrp tDn="uni/fabric/funcprof/podpgrp-bgpRRPodGrp"/>
  </fabricPodS>
</fabricPodP>
```

Configuring BGP External Routed Network Using the REST API

Before You Begin

The tenant where you configure the BGP external routed network is already created.

Procedure

The following shows how to configure the BGP external routed network using the REST API:

Example:

```
<l3extOut descr="" dn="uni/tn-t1/out-l3out-bgp" enforceRtctrl="export" name="l3out-bgp"
```

```

ownerKey="" ownerTag="" targetDscp="unspecified">
<13extRsEctx tnFvCtxName="ctx3"/>
<13extLNodeP configIssues="" descr="" name="13extLNodeP_1" ownerKey="" ownerTag=""
tag="yellow-green" targetDscp="unspecified">
<13extRsNodeL3OutAtt rtrId="1.1.1.1" rtrIdLoopBack="no" tDn="topology/pod-1/node-101"/>
<13extLIfP descr="" name="13extLIfP_2" ownerKey="" ownerTag="" tag="yellow-green">
<13extRsNdIfPol tnNdIfPolName="">
<13extRsIngressQosDppPol tnQosDppPolName="">
<13extRsEgressQosDppPol tnQosDppPolName="">
<13extRsPathL3OutAtt addr="3001::31:0:1:2/120" descr="" encaps="vlan-3001" encapsScope="local"
ifInstT="sub-interface" llAddr="::" mac="00:22:BD:F8:19:FF" mode="regular" mtu="inherit"
tDn="topology/pod-1/paths-101/pathep-[eth1/8]" targetDscp="unspecified">
<bgpPeerP addr="3001::31:0:1:0/120" allowedSelfAsCnt="3" ctrl="send-com,send-ext-com"
descr="" name="" peerCtrl="bfd" privateASctrl="remove-all,remove-exclusive,replace-as"
ttl="1" weight="1000">
<bgpRsPeerPfxPol tnBgpPeerPfxPolName="">
<bgpAsP asn="3001" descr="" name="">
</bgpPeerP>
</13extRsPathL3OutAtt>
</13extLIfP>
<13extLIfP descr="" name="13extLIfP_1" ownerKey="" ownerTag="" tag="yellow-green">
<13extRsNdIfPol tnNdIfPolName="">
<13extRsIngressQosDppPol tnQosDppPolName="">
<13extRsEgressQosDppPol tnQosDppPolName="">
<13extRsPathL3OutAtt addr="31.0.1.2/24" descr="" encaps="vlan-3001" encapsScope="local"
ifInstT="sub-interface" llAddr="::" mac="00:22:BD:F8:19:FF" mode="regular" mtu="inherit"
tDn="topology/pod-1/paths-101/pathep-[eth1/8]" targetDscp="unspecified">
<bgpPeerP addr="31.0.1.0/24" allowedSelfAsCnt="3" ctrl="send-com,send-ext-com" descr=""
name="" peerCtrl="" privateASctrl="remove-all,remove-exclusive,replace-as" ttl="1"
weight="100">
<bgpRsPeerPfxPol tnBgpPeerPfxPolName="">
<bgpLocalAsnP asnPropagate="none" descr="" localAsn="200" name="">
<bgpAsP asn="3001" descr="" name="">
</bgpPeerP>
</13extRsPathL3OutAtt>
</13extLIfP>
</13extLNodeP>
<13extRsL3DomAtt tDn="uni/13dom-13-dom"/>
<13extRsDampeningPol af="ipv6-ucast" tnRtctrlProfileName="damp_rp"/>
<13extRsDampeningPol af="ipv4-ucast" tnRtctrlProfileName="damp_rp"/>
<13extInstP descr="" matchT="AtleastOne" name="13extInstP_1" prio="unspecified"
targetDscp="unspecified">
<13extSubnet aggregate="" descr="" ip="130.130.130.0/24" name="" scope="import-rtctrl">
</13extSubnet>
<13extSubnet aggregate="" descr="" ip="130.130.131.0/24" name="" scope="import-rtctrl"/>
<13extSubnet aggregate="" descr="" ip="120.120.120.120/32" name=""
scope="export-rtctrl,import-security"/>
<13extSubnet aggregate="" descr="" ip="3001::130:130:130:100/120" name=""
scope="import-rtctrl"/>
</13extInstP>
<bgpExtP descr="">
</13extOut>
<rtctrlProfile descr="" dn="uni/tn-t1/prof-damp_rp" name="damp_rp" ownerKey="" ownerTag=""
type="combinable">
<rtctrlCtxP descr="" name="ipv4_rpc" order="0">
<rtctrlScope descr="" name="">
<rtctrlRsScopeToAttrP tnRtctrlAttrPName="act_rule"/>
</rtctrlScope>
</rtctrlCtxP>
</rtctrlProfile>
<rtctrlAttrP descr="" dn="uni/tn-t1/attr-act_rule" name="act_rule">
<rtctrlSetDamp descr="" halfLife="15" maxSuppressTime="60" name="" reuse="750"
suppress="2000" type="dampening-pol"/>
</rtctrlAttrP>

```

Configuring BFD Consumer Protocols Using the REST API

Before You Begin

Procedure

Step 1 The following example shows the interface configuration for bidirectional forwarding detection (BFD):

Example:

```
<fvTenant name="ExampleCorp">
    <bfdIfPol name="bfdIfPol" minTxIntvl="400" minRxIntvl="400" detectMult="5" echoRxIntvl="400"
    echoAdminSt="disabled"/>
    <l3extOut name="l3-out">
        <l3extLNodeP name="leaf1">
            <l3extRsNodeL3OutAtt tDn="topology/pod-1/node-101" rtrId="2.2.2.2"/>

            <l3extLIfP name='portIpv4'>
                <l3extRsPathL3OutAtt tDn="topology/pod-1/paths-101/pathep-[eth1/11]"
                ifInstT='l3-port' addr="10.0.0.1/24" mtu="1500"/>
                <bfdfP type="sha1" key="password">
                    <bfdrIfPol tnBfdIfPolName='bfdIfPol' />
                </bfdfP>
            </l3extLIfP>

        </l3extLNodeP>
    </l3extOut>
</fvTenant>
```

Step 2 The following example shows the interface configuration for enabling BFD on OSPF and EIGRP:

Example:

```
<fvTenant name="ExampleCorp">
    <ospfIfPol name="ospf_intf_pol" cost="10" ctrl="bfd"/>
    <eigrpIfPol ctrl="nh-self,split-horizon,bfd"
    dn="uni/tn-Coke/eigrpIfPol-eigrp_if_default"
    </fvTenant>
```

Step 3 The following example shows the interface configuration for enabling BFD on BGP:

Example:

```
<fvTenant name="ExampleCorp">
    <l3extOut name="l3-out">
        <l3extLNodeP name="leaf1">
            <l3extRsNodeL3OutAtt tDn="topology/pod-1/node-101" rtrId="2.2.2.2"/>

            <l3extLIfP name='portIpv4'>
                <l3extRsPathL3OutAtt tDn="topology/pod-1/paths-101/pathep-[eth1/11]"
                ifInstT='l3-port' addr="10.0.0.1/24" mtu="1500">
                    <bgpPeerP addr="4.4.4.4/24" allowedSelfAsCnt="3" ctrl="bfd" descr=""
                    name="" peerCtrl="" ttl="1">
                        <bgpRsPeerPfxPol tnBgpPeerPfxPolName="" />
                        <bgpAsP asn="3" descr="" name="" />
                    </bgpPeerP>
                </l3extRsPathL3OutAtt>
            </l3extLIfP>

        </l3extLNodeP>
    </l3extOut>
</fvTenant>
```

```

        </l3extOut>
    </fvTenant>
Step 4 The following example shows the interface configuration for enabling BFD on Static Routes:

Example:

<fvTenant name="ExampleCorp">
    <l3extOut name="l3-out">
        <l3extLNodeP name="leaf1">
            <l3extRsNodeL3OutAtt tDn="topology/pod-1/node-101" rtrId="2.2.2.2">
                <ipRouteP ip="192.168.3.4" rtCtrl="bfd">
                    <ipNexthopP nhAddr="192.168.62.2"/>
                </ipRouteP>
            </l3extRsNodeL3OutAtt>
            <l3extLIfP name='portIpv4'>
                <l3extRsPathL3OutAtt tDn="topology/pod-1/paths-101/pathep-[eth1/3]">
                    ifInstT='l3-port' addr="10.10.10.2/24" mtu="1500" status="created,modified" />
                </l3extLIfP>
            </l3extLNodeP>
        </l3extOut>
    </fvTenant>

```

Configuring BFD Globally Using the REST API

Before You Begin

Procedure

The following REST API shows the global configuration for bidirectional forwarding detection (BFD):

Example:

```

<polUni>
    <infraInfra>
        <bfdIpv4InstPol name="default" echoSrcAddr="1.2.3.4" slowIntvl="1000" minTxIntvl="150" minRxIntvl="250" detectMult="5" echoRxIntvl="200"/>
        <bfdIpv6InstPol name="default" echoSrcAddr="34::1/64" slowIntvl="1000" minTxIntvl="150" minRxIntvl="250" detectMult="5" echoRxIntvl="200"/>
    </infraInfra>
</polUni>

```

Configuring BFD Interface Override Using the REST API

Before You Begin

Procedure

The following REST API shows the interface override configuration for bidirectional forwarding detection (BFD):

Example:

```
<fvTenant name="ExampleCorp">
```

```

<bfdIfPol name="bfdIfPol" minTxIntvl="400" minRxIntvl="400" detectMult="5" echoRxIntvl="400"
echoAdminSt="disabled"/>
<l3extOut name="l3-out">
    <l3extLNodeP name="leaf1">
        <l3extRsNodeL3OutAtt tDn="topology/pod-1/node-101" rtrId="2.2.2.2"/>
        <l3extLIfP name='portIpv4'>
            <l3extRsPathL3OutAtt tDn="topology/pod-1/paths-101/pathp-[eth1/11]"
ifInstT='l3-port' addr="10.0.0.1/24" mtu="1500"/>
            <bfdIfP type="sha1" key="password">
                <bfdRsIfPol tnBfdIfPolName='bfdIfPol' />
            </bfdIfP>
        </l3extLIfP>
    </l3extLNodeP>
</l3extOut>
</fvTenant>

```

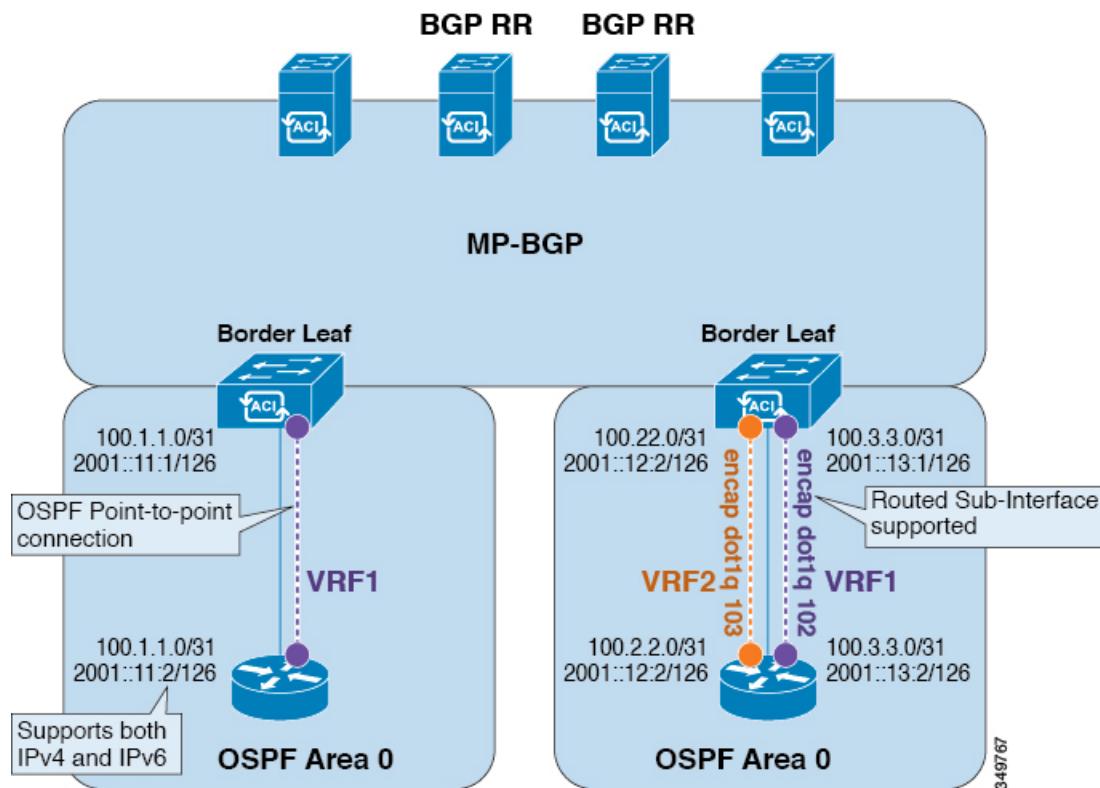
OSPF

OSPF Layer 3 Outside Connections

OSPF Layer 3 Outside connections can be normal or NSSA areas. The backbone (area 0) area is also supported as an OSPF Layer 3 Outside connection area. ACI supports both OSPFv2 for IPv4 and OSPFv3 for IPv6. When creating an OSPF Layer 3 Outside, it is not necessary to configure the OSPF version. The correct OSPF process is created automatically based on the interface profile configuration (IPv4 or IPv6 addressing). Both IPv4 and IPv6 protocols are supported on the same interface (dual stack) but it is necessary to create two separate interface profiles.

Layer 3 Outside connections are supported for the routed interfaces, routed sub-interfaces, and SVIs. The SVIs are used when there is a need to share the physical connect for both L2 and L3 traffic. The SVIs are supported on ports, port-channels, and VPC port-channels.

Figure 35: OSPF Layer3 Out Connections



When an SVI is used for an Layer 3 Outside connection, an external bridge domain is created on the border leaf switches. The external bridge domain allows connectivity between the two VPC switches across the ACI fabric. This allows both the VPC switches to establish the OSPF adjacencies with each other and the external OSPF device.

When running OSPF over a broadcast network, the time to detect a failed neighbor is the dead time interval (default 40 seconds). Reestablishing the neighbor adjacencies after a failure may also take longer due to designated router (DR) election.



Note A link or port-channel failure to one VPC Node does not cause an OSPF adjacency to go down. The OSPF adjacency can stay up via the external BD accessible through the other VPC node.

Creating OSPF External Routed Network for Management Tenant Using REST API

- You must verify that the router ID and the logical interface profile IP address are different and do not overlap.

- The following steps are for creating an OSPF external routed network for a management tenant. To create an OSPF external routed network for a tenant, you must choose a tenant and create a VRF for the tenant.
- For more details, see also the KB article about *Transit Routing*.

Procedure

Create an OSPF external routed network for management tenant.

Example:

POST: <https://apic-ip-address/api/mo/uni/tn-mgmt.xml>

```
<fvTenant name="mgmt">
  <fvBD name="bd1">
    <fvRsBDToOut tnL3extOutName="RtdOut" />
    <fvSubnet ip="1.1.1.1/16" />
    <fvSubnet ip="1.2.1.1/16" />
    <fvSubnet ip="40.1.1.1/24" scope="public" />
      <fvRsCtx tnFvCtxName="inb" />
    </fvBD>
    <fvCtx name="inb" />

    <l3extOut name="RtdOut">
      <l3extRsL3DomAtt tDn="uni/l3dom-extdom"/>
      <l3extInstP name="extMgmt">
        </l3extInstP>
        <l3extLNodeP name="borderLeaf">
          <l3extRsNodeL3OutAtt tDn="topology/pod-1/node-101" rtrId="10.10.10.10"/>
          <l3extRsNodeL3OutAtt tDn="topology/pod-1/node-102" rtrId="10.10.10.11"/>
            <l3extLIfP name='portProfile'>
              <l3extRsPathL3OutAtt tDn="topology/pod-1/paths-101/pathep-[eth1/40]" ifInstT='l3-port' addr="192.168.62.1/24"/>
              <l3extRsPathL3OutAtt tDn="topology/pod-1/paths-102/pathep-[eth1/40]" ifInstT='l3-port' addr="192.168.62.5/24"/>
                <ospfIfP/>
              </l3extLIfP>
            </l3extLNodeP>
            <l3extRsEctx tnFvCtxName="inb"/>
              <ospfExtP areaId="57" />
            </l3extOut>
    </fvTenant>
```

EIGRP

Overview

This article provides a typical example of how to configure Enhanced Interior Gateway Routing Protocol (EIGRP) when using the Cisco APIC. The following information applies when configuring EIGRP:

- The tenant, VRF, and bridge domain must already be created.
- The Layer 3 outside tenant network must already be configured.
- The route control profile under routed outside must already be configured.
- The EIGRP VRF policy is the same as the EIGRP family context policy.
- EIGRP supports only export route control profile. The configuration related to route controls is common across all the protocols.

You can configure EIGRP to perform automatic summarization of subnet routes (route summarization) into network-level routes. For example, you can configure subnet 131.108.1.0 to be advertised as 131.108.0.0 over interfaces that have subnets of 192.31.7.0 configured. Automatic summarization is performed when there are two or more network router configuration commands configured for the EIGRP process. By default, this feature is enabled.

For more information about route summarization, see the *Cisco Application Centric Infrastructure Fundamentals Guide*.

Configuring EIGRP Using the REST API

Procedure

Step 1 Configure an EIGRP context policy.

Example:

```
<polUni>
    <fvTenant name="cisco_6">
        <eigrpCtxAfPol actIntvl="3" descr="" dn="uni/tn-cisco_6/eigrpCtxAfP-eigrp_default_pol"
        extDist="170"
            intDist="90" maxPaths="8" metricStyle="narrow" name="eigrp_default_pol" ownerKey=""
            ownerTag="" />
        </fvTenant>
    </polUni>
```

Step 2 Configure an EIGRP interface policy.

Example:

```
<polUni>
    <fvTenant name="cisco_6">
        <eigrpIfPol bw="10" ctrl="nh-self,split-horizon" delay="10" delayUnit="tens-of-micro"
        descr="" dn="uni/tn-cisco_6/eigrpIfPol-eigrp_if_default"
            helloIntvl="5" holdIntvl="15" name="eigrp_if_default" ownerKey="" ownerTag="" />
    </fvTenant>
</polUni>
```

Step 3 Configure an EIGRP VRF.

Example:

IPv4:

```
<polUni>
    <fvTenant name="cisco_6">
        <fvCtx name="dev">
            <fvRsCtxToEigrpCtxAfPol tnEigrpCtxAfPolName="eigrp_ctx_pol_v4" af="1"/>
        </fvCtx>
    </fvTenant>
</polUni>
```

IPv6:

```
<polUni>
    <fvTenant name="cisco_6">
        <fvCtx name="dev">
            <fvRsCtxToEigrpCtxAfPol tnEigrpCtxAfPolName="eigrp_ctx_pol_v6" af="ipv6-ucast"/>
        </fvCtx>
    </fvTenant>
</polUni>
```

Step 4 Configure an EIGRP Layer3 Outside.

Example:**IPv4**

```
<polUni>
    <fvTenant name="cisco_6">
        <l3extOut name="ext">
            <eigrpExtP asn="4001"/>
            <l3extLNodeP name="node1">
                <l3extLIfP name="intf_v4">
                    <l3extRsPathL3OutAtt addr="201.1.1.1/24" ifInstT="l3-port"
                        tDn="topology/pod-1/paths-101/pathep-[eth1/4]" />
                    <eigrpIfP name="eigrp_ifp_v4">
                        <eigrpRsIfPol tnEigrpIfPolName="eigrp_if_pol_v4"/>
                    </eigrpIfP>
                </l3extLIfP>
            </l3extLNodeP>
        </l3extOut>
    </fvTenant>
</polUni>
```

IPv6

```
<polUni>
    <fvTenant name="cisco_6">
        <l3extOut name="ext">
            <eigrpExtP asn="4001"/>
            <l3extLNodeP name="node1">
                <l3extLIfP name="intf_v6">
                    <l3extRsPathL3OutAtt addr="2001::1/64" ifInstT="l3-port"
                        tDn="topology/pod-1/paths-101/pathep-[eth1/4]" />
                    <eigrpIfP name="eigrp_ifp_v6">
                        <eigrpRsIfPol tnEigrpIfPolName="eigrp_if_pol_v6"/>
                    </eigrpIfP>
                </l3extLIfP>
            </l3extLNodeP>
        </l3extOut>
    </fvTenant>
</polUni>
```

IPv4 and IPv6

```
<polUni>
    <fvTenant name="cisco_6">
        <l3extOut name="ext">
            <eigrpExtP asn="4001"/>
            <l3extLNodeP name="node1">
                <l3extLIfP name="intf_v4">
                    <l3extRsPathL3OutAtt addr="201.1.1.1/24" ifInstT="l3-port"
                        tDn="topology/pod-1/paths-101/pathep-[eth1/4]" />
                    <eigrpIfP name="eigrp_ifp_v4">
                        <eigrpRsIfPol tnEigrpIfPolName="eigrp_if_pol_v4"/>
                    </eigrpIfP>
                </l3extLIfP>

                <l3extLIfP name="intf_v6">
                    <l3extRsPathL3OutAtt addr="2001::1/64" ifInstT="l3-port"
                        tDn="topology/pod-1/paths-101/pathep-[eth1/4]" />
                    <eigrpIfP name="eigrp_ifp_v6">
                        <eigrpRsIfPol tnEigrpIfPolName="eigrp_if_pol_v6"/>
                    </eigrpIfP>
                </l3extLIfP>
            </l3extLNodeP>
        </l3extOut>
    </fvTenant>
</polUni>
```

Step 5 (Optional) Configure the interface policy knobs.

Example:

```
<polUni>
    <fvTenant name="cisco_6">
        <eigrpIfPol bw="1000000" ctrl="nh-self,split-horizon" delay="10"
                     delayUnit="tens-of-micro" helloIntvl="5" holdIntvl="15" name="default"/>
    </fvTenant>
</polUni>
```

The `bandwidth (bw)` attribute is defined in Kbps. The `delayUnit` attribute can be "tens of micro" or "pico".

Neighbor Discovery

Neighbor Discovery

The IPv6 Neighbor Discovery (ND) protocol is responsible for address autoconfiguration of nodes, discovery of other nodes on the link, determining the link-layer addresses of other nodes, duplicate address detection, finding available routers and DNS servers, address prefix discovery, and maintaining reachability information about the paths to other active neighbor nodes.

ND-specific Neighbor Solicitation/Neighbor Advertisement (NS/NA) and Router Solicitation/Router Advertisement (RS/RA) packet types are supported on all ACI fabric Layer 3 interfaces, including physical, L3 Sub-if, and SVI (external and pervasive). RS/RA packets are used for autoconfiguration for all L3 interfaces but are only configurable for pervasive SVIs. ACI bridge domain ND always operates in flood mode; unicast mode is not supported.

The ACI fabric ND support includes the following:

- Interface policies (`nd:IfPol`) control ND timers and behavior for NS/NA messages.
- ND prefix policies (`nd:PfxPol`) controls RA messages.
- Configuration of IPv6 subnets for ND (`fv:Subnet`).
- ND interface policies for external networks.
- Configurable ND subnets for external networks, and arbitrary subnet configurations for pervasive bridge domains are not supported.

Configuration options include the following:

- Adjacencies
 - Configurable Static Adjacencies : (<vrf, L3Iface, ipv6 address> --> mac address)
 - Dynamic Adjacencies : Learnt via exchange of NS/NA packets
- Per Interface
 - Control of ND packets (NS/NA)
 - Neighbor Solicitation Interval
 - Neighbor Solicitation Retry count

- Control of RA packets
 - Suppress RA
 - Suppress RA MTU
 - RA Interval, RA Interval minimum, Retransmit time
- Per Prefix (advertised in RAs) control
 - Lifetime, preferred lifetime
 - Prefix Control (autoconfiguration, on link)

Creating the Tenant, VRF, and Bridge Domain with IPv6 Neighbor Discovery Using the REST API

Procedure

Create a tenant, VRF, bridge domain with a neighbor discovery interface policy and a neighbor discovery prefix policy.

Example:

```
<fvTenant descr="" dn="uni/tn-ExampleCorp" name="ExampleCorp" ownerKey="" ownerTag="">
  <ndIfPol name="NDPo1001" ctrl="managed-cfg" descr="" hopLimit="64" mtu="1500"
    nsIntvl="1000" nsRetries="3" ownerKey="" ownerTag="" raIntvl="600" raLifetime="1800"
    reachableTime="0" retransTimer="0"/>
    <fvCtx descr="" knwMcastAct="permit" name="pvn1" ownerKey="" ownerTag=""
      pcEnfPref="enforced">
      </fvCtx>
      <fvBD arpFlood="no" descr="" mac="00:22:BD:F8:19:FF" multiDstPktAct="bd-flood" name="bd1"
        ownerKey="" ownerTag="" unicastRoute="yes" unkMacUcastAct="proxy" unkMcastAct="flood">
        <fvRsBDToNdP tnNdIfPolName="NDPo1001"/>
        <fvRsCtx tnfVctxName="pvn1"/>
        <fvSubnet ctrl="nd" descr="" ip="34::1/64" name="" preferred="no" scope="private">
          <fvRsNdPfxPol tnNdPfxPolName="NDPfxPo1001"/>
        </fvSubnet>
        <fvSubnet ctrl="nd" descr="" ip="33::1/64" name="" preferred="no" scope="private">
          <fvRsNdPfxPol tnNdPfxPolName="NDPfxPo1002"/>
        </fvSubnet>
      </fvBD>
      <ndPfxPol ctrl="auto-cfg, on-link" descr="" lifetime="1000" name="NDPfxPo1001" ownerKey=""
        ownerTag="" prefLifetime="1000"/>
        <ndPfxPol ctrl="auto-cfg, on-link" descr="" lifetime="4294967295" name="NDPfxPo1002"
          ownerKey="" ownerTag="" prefLifetime="4294967295"/>
    </fvTenant>
```

Note If you have a public subnet when you configure the routed outside, you must associate the bridge domain with the outside configuration.



CHAPTER 14

Managing Layer 4 to Layer 7 Services

- [About Layer 4 to Layer 7 Services, page 279](#)
- [Access for Managing Layer 4 to Layer 7 Services, page 280](#)
- [Device Packages, page 283](#)
- [Trunking, page 284](#)
- [Device Selection Policies, page 285](#)
- [Service Graph Templates, page 286](#)
- [Layer 4 to Layer 7 Parameters, page 288](#)
- [Copy Services, page 291](#)
- [Developing Automation, page 293](#)
- [Example: Configuring Layer 4 to Layer 7 Services \(Firewall\), page 300](#)
- [Example: Configuring Layer 4 to Layer 7 Route Peering, page 309](#)

About Layer 4 to Layer 7 Services

About Application-Centric Infrastructure Layer 4 to Layer 7 Services

Although VLAN and virtual routing and forwarding (VRF) stitching is supported by traditional service insertion models, the Application Policy Infrastructure Controller (APIC) can automate service insertion while acting as a central point of policy control. The APIC policies manage both the network fabric and services appliances. The APIC can configure the network automatically so that traffic flows through the services. The APIC can also automatically configure the service according to the application's requirements, which allows organizations to automate service insertion and eliminate the challenge of managing the complex techniques of traditional service insertion.

Before you begin, the following APIC objects must be configured:

- The tenant that will provide/consume the Layer 4 to Layer 7 services
- A Layer 3 outside network for the tenant

- At least one bridge domain
- An application profile
- A physical domain or a VMM domain

For a VMM domain, configure VMM domain credentials and configure a vCenter/vShield controller profile.

- A VLAN pool with an encapsulation block range
- At least one contract
- At least one EPG

You must perform the following tasks to deploy Layer 4 to Layer 7 services:

1 Import a Device Package .

Only the provider administrator can import the device package.

2 Register the device and the logical interfaces.

This task also registers concrete devices and concrete interfaces, and configures concrete device parameters.

3 Create a Logical Device.

4 Configure device parameters.

5 Optional. If you are configuring an ASA Firewall service, enable trunking on the device.

6 Configure a Device Selection Policy.

7 Configure a Service Graph Template.

a Select the default service graph template parameters from an application profile.

b Configure additional service graph template parameters, if needed.

8 Attach the service graph template to a contract.

9 Configure additional configuration parameters, if needed.

For more information about deploying Layer 4 to Layer 7 services, see the *Cisco APIC Layer 4 to Layer 7 Services Deployment Guide*.

Access for Managing Layer 4 to Layer 7 Services

Configure In-Band Connectivity to Devices Using Tenant's VRF Using the REST API

The following is an example of using REST APIs to configure in-band connectivity to devices using tenant's VRF:

1 Define the EPG that is to be used for management.



Note Ensure to open up the ports to the domain mappings using the appropriate selectors configuration.

In the following, the EPG "services" is used for the management of the Services Devices/VMs subnet that is used for Tenant vrf devicemanagement (3.3.3.0/24).

```
<polUni>
  <fvTenant name="tenant1">
    <fvCtx name="mgmt_ctx1"/>
    <vnsCtrlrMgmtPol ctxDn="uni/tn-tenant1/ctx-mgmt_ctx1">
      <vnsRsMgmtAddr tDn="uni/tn-tenant1/ap-services/epg-ifc/CtrlrAddrInst-ifc"/>
    </vnsCtrlrMgmtPol>
    <fvBD name="mgmt_ServicesMgmtBD">
      <fvRsCtx tnFvCtxName="mgmt_ctx1"/>
      <fvSubnet ip="3.3.3.3/24"/>
    </fvBD>
    <fvAp name="services">
      <fvAEPg name="ifc">
        <fvRsBd tnFvBDName="mgmt_ServicesMgmtBD"/>
        <vnsAddrInst name="ifc">
          <fvnsUcastAddrBlk from="3.3.3.100/24" to="3.3.3.200/24"/>
        </vnsAddrInst>
        <fvRsDomAtt tDn="uni/vmmp-VMware/dom-mininet"/>
      </fvAEPg>
    </fvAp>
  </fvTenant>
</polUni>
```

2 Associate the EPG to the LDevVip.

```
<polUni>
  <fvTenant name="tenant1">

    <vnsLDevVip name="ADCCluster1"
      funcType="GoTo" devtype="VIRTUAL">
      <vnsRsMDevAtt tDn="uni/infra/mDev-Citrix-NetScaler-10.5"/>
      <vnsRsALDevToDomP tDn="uni/vmmp-VMware/dom-mininet"/>
      <vnsRsDevEpg tDn="uni/tn-tenant1/ap-services/epg-ifc"/>

      <vnsCMgmt name="devMgmt"
        host="3.3.3.180"
        port="80"/>

      <vnsCCred name="username"
        value="nsroot"/>

      <vnsCCredSecret name="password"
        value="nsroot"/>
    </vnsLDevVip>
  </fvTenant>
</polUni>
```

Configuring In-Band Connectivity to Devices Using Management Tenant VRF Using the REST API

The following is an example of using REST APIs to configure in-band connectivity to devices using management tenant VRF:

1 Create an EPG l4l7MgmtEpg in tenant management.



Note

l4l7MgmtEpg is a part of bd access which is under inb context in tn-mgmt.

contract1 is the contract between the tn-mgmt l4l7MgmtEpg and tn-mgmt inb default EPG.

```
<polUni>
  <fvTenant dn="uni/tn-mgmt">
```

```

<fvAp name="services">
    <fvAEPg name="l4l7MgmtEpg">
        <fvRsBd tnFvBDName="access" />
            <fvRsDomAtt tDn="uni/vmmp-VMware/dom-mininet" />
        <fvRsCons tnVzBrCPName='contract1'>
            </fvRsCons>
        </fvAEPg>
    </fvAp>
    <fvBD name="access">
        <fvSubnet ip="3.3.3.3/24" />
        <fvRsCtx tnFvCtxName="inb"/>
    </fvBD>
    <vzFilter name='all'>
        <vzEntry name='all' ></vzEntry>
    </vzFilter>
    <vzBrCP name="contract1" scope="tenant">
        <vzSubj name='subj1'>
            <vzInTerm>
                <vzRsFiltAtt tnVzFilterName="all" />
            </vzInTerm>
            <vzOutTerm>
                <vzRsFiltAtt tnVzFilterName="all" />
            </vzOutTerm>
        </vzSubj>
    </vzBrCP>
    </fvTenant>
</polUni>

```

2 Ensure that the Service Device/VM has the mgmt IP address in the subnet 3.3.3.0/24.

This is the same subnet that tn-mgmt access BD has been configured with. (See configuration in earlier step.)

3 Add the following to the LDevVip:



This points to the EPG that was created in the earlier step

```
<vnsRsDevEpg tDn="uni/tn-mgmt/ap-services/epg-l4l7MgmtEpg"/>
```

```

<polUni>
    <fvTenant name="mgmt">

        <vnsLDevVip name="ADCCluster1"
            funcType="GoTo" devtype="VIRTUAL">
            <vnsRsMDevAtt tDn="uni/infra/mDev-Citrix-NetScaler-10.5"/>
            <vnsRsALDevToDomP tDn="uni/vmmp-VMware/dom-mininet"/>
            <vnsRsDevEpg tDn="uni/tn-mgmt/ap-services/epg-l4l7MgmtEpg"/>

            <vnsCMgmt name="devMgmt"
                host="3.3.3.180"
                port="80"/>

            <vnsCCred name="username"
                value="nsroot"/>

            <vnsCCredSecret name="password"
                value="nsroot"/>
        </vnsLDevVip>
    </fvTenant>
</polUni>

```

4 Add the route in service Device/VM to point to the IFC inband gateway.

For example, on the route on netScaler, add route 3.0.0.0 255.255.255.0 3.3.3.3, where 3.0.0.0/24 is the IFC inband subnet and 3.3.3.3 is the SVI IP for l4l7MgmtEpg.

5 Verify the following:

- The route table on IFC has an entry for ifc inband IP.
- The IFC can ping the l4l7MgmtEpg gateway on the leaf.
- The service node can ping the l4l7MgmtEpg SVI gateway and IFC inb SVI Ip.

Device Packages

About the Device Package

The Application Policy Infrastructure Controller (APIC) requires a device package to configure and monitor service devices. A device package manages a single class of service devices and provides the APIC with information about the device and its capabilities.

For more information about device packages, see the *Cisco APIC Layer 4 to Layer 7 Device Package Development Guide*.

Notes for Installing a Device Package with REST

- A device package can be installed using an HTTP or HTTPS POST.
- If HTTP is enabled on APIC, the URL for the POST is "http://10.10.10.10/ppi/node/mo/.xml".
- If HTTPS is enabled on APIC, the URL for the POST is "https://10.10.10.10/ppi/node/mo/.xml".
- The message must have a valid session cookie.
- The body of the POST should contain the device package being uploaded. Only one package is allowed in a POST.

Uploading a Device Package File Using the API

To install a service device, you must upload a device package file to APIC. The API command for this operation uses a special form of URI:

```
{http| https}://host[:port]/ppi/node/mo/.{json| xml}
```

The URI path contains 'ppi' (package programming interface) instead of 'api', and the command is sent as a POST operation with the device package file as the body of the message. The device package file is a zip file.

This example shows an API operation that uploads a device package file:

```
POST https://192.0.20.123/ppi/node/mo/.json
```

For more information about installing L4-L7 service device packages, see *Cisco APIC Layer 4 to Layer 7 Services Deployment Guide*.

Installing a Device Package Using the REST API

You can install a device package using an HTTP or HTTPS POST.

Procedure

Install the device package.

- If HTTP is enabled on the Application Policy Infrastructure Controller (APIC) , the URL for the POST is as follows:

`http://10.10.10.10/ppi/node/mo/.xml`

- If HTTPS is enabled on the APIC, the URL for the POST is as follows:

`https://10.10.10.10/ppi/node/mo/.xml`

The message must have a valid session cookie.

The body of the POST should contain the device package being uploaded. Only one package is allowed in a POST.

Using an Imported Device with the REST APIs

The following REST API uses an imported device:

```
<polUni>
  <fvTenant dn="uni/tn-tenant1" name="tenant1">
    <vnslDevIf ldev="uni/tn-mgmt/lDevVip-ADCCluster1"/>
    <vnslDevCtx ctrctNameOrLbl="any" graphNameOrLbl="any" nodeNameOrLbl="any">
      <vnsLDevCtxToLDev tDn="uni/tn-tenant1/lDevIf-[uni/tn-mgmt/lDevVip-ADCCluster1]"/>
      <vnsLifCtx connNameOrLbl="inside">
        <vnsRsLifCtxToLif
          tDn="uni/tn-tenant1/lDevIf-[uni/tn-mgmt/lDevVip-ADCCluster1]/lDevIfLif-inside"/>
        <fvSubnet ip="10.10.10.10/24"/>
        <vnsRsLifCtxToBD tDn="uni/tn-tenant1/BD-tenant1BD1"/>
      </vnsLifCtx>
      <vnsLifCtx connNameOrLbl="outside">
        <vnsRsLifCtxToLif
          tDn="uni/tn-tenant1/lDevIf-[uni/tn-mgmt/lDevVip-ADCCluster1]/lDevIfLif-outside"/>
        <fvSubnet ip="70.70.70.70/24"/>
        <vnsRsLifCtxToBD tDn="uni/tn-tenant1/BD-tenant1BD4"/>
      </vnsLifCtx>
    </vnsLDevCtx>
  </fvTenant>
</polUni>
```

Trunking

About Trunking

You can enable trunking for a Layer 4 to Layer 7 virtual ASA device, which uses trunk port groups to aggregate the traffic of endpoint groups. Without trunking, a virtual service device can have only 1 VLAN per interface and up to 10 service graphs. With trunking enabled, the virtual service device can have an unlimited number of service graphs.

For more information about trunk port groups, see the *Cisco ACI Virtualization Guide*.

Trunking is supported only on a virtual ASA device. The ASA device package must be version 1.2.7.8 or later.

Enabling Trunking on a Layer 4 to Layer 7 Virtual ASA device Using the REST APIs

The following procedure provides an example of enabling trunking on a Layer 4 to Layer 7 virtual ASA device using the REST APIs.

Before You Begin

- You must have configured a Layer 4 to Layer 7 virtual ASA device.

Procedure

Enable trunking on the Layer 4 to Layer 7 device named `InsiemeCluster`:

```
<polUni>
    <fvTenant name="tenant1">
        <vnsLDevVip name="InsiemeCluster" devtype="VIRTUAL" trunking="yes">
            ...
            ...
        </vnsLDevVip>
    </fvTenant>
</polUni>
```

Device Selection Policies

About Device Selection Policies

A device can be selected based on a contract name, a graph name, or the function node name inside the graph. After you create a device, you can create a device context, which provides a selection criteria policy for a device.

A device selection policy (also known as a device context) specifies the policy for selecting a device for a service graph template. This allows an administrator to have multiple device and then be able to use them for different service graph templates. For example, an administrator can have a device that has high-performance ADC appliances and another device that has lower-performance ADC appliances. Using two different device selection policies, one for the high-performance ADC device and the other for the low-performance ADC device, the administrator can select the high-performance ADC device for the applications that require higher performance and select the low-performance ADC devices for the applications that require lower performance.

Creating a Device Selection Policy Using the REST API

The following REST API creates a device selection policy:

```
<polUni>
    <fvTenant dn="uni/tn-acme" name="acme">
        <vnsLDevCtx ctrctNameOrLbl="webCtrct" graphNameOrLbl="G1" nodeNameOrLbl="Node1">
            <vnsRsLDevCtxToLDev tDn="uni/tn-acme/lDevVip-ADCCluster1"/>
        <!-- The connector name C4, C5, etc.. should match the
            Function connector name used in the service graph template -->
```

```

<vnsLIfCtx connNameOrLbl="C4">
    <vnsRsLIfCtxToLIf tDn="uni/tn-acme/lDevVip-ADCCluster1/LIf-ext"/>
</vnsLIfCtx>
<vnsLIfCtx connNameOrLbl="C5">
    <vnsRsLIfCtxToLIf tDn="uni/tn-acme/lDevVip-ADCCluster1/LIf-int"/>
</vnsLIfCtx>
</vnsLDevCtx>
</fvTenant>
</polUni>

```

Adding a Logical Interface in a Device Using the REST APIs

The following REST API adds a logical interface in a device:

```

<polUni>
    <fvTenant dn="uni/tn-acme" name="acme">
        <vnsLDevVip name="ADCCluster1">

            <!-- The LIF name defined here (such as e.g., ext, or int) should match the
                vnsRsLIfCtxToLIf 'tDn' defined in LifCtx -->

            <vnsLIf name="ext">
                <vnsRsMetaIf tDn="uni/infra/mDev-Acme-ADC-1.0/mIfLbl-outside"/>
                <vnsRsCIfAtt tDn="uni/tn-acme/lDevVip-ADCCluster1/cDev-ADC1/cIf-ext"/>
            </vnsLIf>
            <vnsLIf name="int">
                <vnsRsMetaIf tDn="uni/infra/mDev-Acme-ADC-1.0/mIfLbl-inside"/>
                <vnsRsCIfAtt tDn="uni/tn-acme/lDevVip-ADCCluster1/cDev-ADC1/cIf-int"/>
            </vnsLIf>
        </vnsLDevVip>
    </fvTenant>
</polUni>

```

Service Graph Templates

About Service Graph Templates

The Cisco Application Centric Infrastructure (ACI) allows you to define a sequence of meta-devices, such as a firewall of a certain type followed by a load balancer of a certain make and version. This is called a service graph template, also known as a abstract graph. When a service graph template is referenced by a contract, the service graph template is instantiated by mapping it to concrete devices, such as the firewall and load balancers that are present in the fabric. The mapping happens with the concept of a "context". The "device context" is the mapping configuration that allows the ACI to identify which firewalls and which load balancers can be mapped to the service graph template. Another key concept is the "logical device", which represents the cluster of concrete devices. The rendering of the service graph template is based on identifying the suitable logical devices that can be inserted in the path that is defined by a contract.

The ACI treats services as an integral part of an application. Any services that are required are treated as a service graph that is instantiated on the ACI fabric from the Cisco Application Policy Infrastructure Controller (APIC). Users define the service for the application, while service graph templates identify the set of network or service functions that are needed by the application. Once the graph is configured in the APIC, the APIC automatically configures the services according to the service function requirements that are specified in the service graph template. The APIC also automatically configures the network according to the needs of the service function that is specified in the service graph template, which does not require any change in the service device.

Configuring a Service Graph Template Using the REST APIs

You can configure a service graph template using the following REST API:

```
<polUni>
    <fvTenant dn="uni/tn-acme" name="acme">
        <!--L3 Network-->
        <fvCtx name="MyNetwork"/>
        <!-- Bridge Domain for MySrvr EPG -->
        <fvBD name="MySrvrBD">
            <fvRsCtx tnFvCtxName="MyNetwork" />
            <fvSubnet ip="10.10.10.10/24">
                </fvSubnet>
            </fvBD>
            <!-- Bridge Domain for MyClnt EPG -->
            <fvBD name="MyClntBD">
                <fvRsCtx tnFvCtxName="MyNetwork" />
                <fvSubnet ip="20.20.20.20/24">
                    </fvSubnet>
                </fvBD>
            <fvAp dn="uni/tn-acme/ap-MyAP" name="MyAP">
                <fvAEPg dn="uni/tn-acme/ap-MyAP/epg-MyClnt" name="MyClnt">
                    <fvRsBd tnFvBDName="MySrvrBD" />
                    <fvRsDomAtt tDn="uni/vmmp-Vendor1/dom-MyVMs" />
                    <fvRsProv tnVzBrCPName="webCtrct">
                        </fvRsProv>
                    <fvRsPathAtt tDn="topology/pod-1/paths-17/pathep-[eth1/21]" encap="vlan-202"/>
                    <fvRsPathAtt tDn="topology/pod-1/paths-18/pathep-[eth1/21]" encap="vlan-202"/>
                </fvAEPg>
                <fvAEPg dn="uni/tn-acme/ap-MyAP/epg-MySRVR" name="MySRVR">
                    <fvRsBd tnFvBDName="MyClntBD" />
                    <fvRsDomAtt tDn="uni/vmmp-Vendor1/dom-MyVMs" />
                    <fvRsCons tnVzBrCPName="webCtrct">
                        </fvRsCons>
                    <fvRsPathAtt tDn="topology/pod-1/paths-17/pathep-[eth1/21]" encap="vlan-203"/>
                    <fvRsPathAtt tDn="topology/pod-1/paths-18/pathep-[eth1/21]" encap="vlan-203"/>
                </fvAEPg>
            </fvAp>
        </fvTenant>
    </polUni>
```

Creating a Security Policy Using the REST APIs

You can create a security policy using the following REST API:

```
<polUni>
    <fvTenant dn="uni/tn-acme" name="acme">
        <vzFilter name="HttpIn">
            <vzEntry name="e1" prot="6" dToPort="80"/>
        </vzFilter>
        <vzBrCP name="webCtrct">
            <vzSubj name="http">
                <vzRsSubjFiltAtt tnVzFilterName="HttpIn"/>
            </vzSubj>
        </vzBrCP>
    </fvTenant>
</polUni>
```

Layer 4 to Layer 7 Parameters

About Modifying the Configuration Parameters of a Deployed Service Graph

When you first deploy a service graph, the configuration parameters or functions for the service graph must be defined before you can successfully deploy the service graph. These configuration parameters or functions include device network configurations, such as IP addresses, route prefix, and next hop information, as well as the services configuration, such as the IP access list for a firewall or server load balancing configuration for a load balancer.

You must modify the service graph function as part of the day-to-day operation of the Application Policy Infrastructure Controller (APIC). You can modify a service graph's configuration parameters and functions by using the GUI or CLI of the APIC. Modifying functions of a service device through the APIC does not require changes on a service device.

Example XML POST for an Application EPG With Configuration Parameters

The following XML example shows configuration parameters inside of the device package:

```
<fvAEPg dn="uni/tn-acme/ap-myApp/epg-app" name="app">
  <vnsFolderInst ctrctNameOrLbl="any" graphNameOrLbl="any" nodeNameOrLbl="any" key="Monitor">
    name="monitor1">
      <vnsRsFolderInstToMFolder tDn="uni/infra/mDev-Acme-ADC-1.0/mDevCfg/mFolder-Monitor"/>
      <vnsParamInst name="weight" key="weight" value="10"/>
    </vnsFolderInst>

    <vnsFolderInst ctrctNameOrLbl="any" graphNameOrLbl="any" nodeNameOrLbl="any" key="Service">
      name="Service1">
        <vnsParamInst name="servicename" key="servicename" value="crpvgrtst02-8010"/>
        <vnsParamInst name="servicetype" key="servicetype" value="TCP"/>
        <vnsParamInst name="servername" key="servername" value="s192.168.100.100"/>
        <vnsParamInst name="serveripaddress" key="serveripaddress" value="192.168.100.100"/>
        <vnsParamInst name="serviceport" key="serviceport" value="8080"/>
        <vnsParamInst name="svrtimeout" key="svrtimeout" value="9000" />
        <vnsParamInst name="clttimeout" key="clttimeout" value="9000" />
        <vnsParamInst name="usip" key="usip" value="NO" />
        <vnsParamInst name="useproxyport" key="useproxyport" value="" />
        <vnsParamInst name="cip" key="cip" value="ENABLED" />
        <vnsParamInst name="cka" key="cka" value="NO" />
        <vnsParamInst name="sp" key="sp" value="OFF" />
        <vnsParamInst name="cmp" key="cmp" value="NO" />
        <vnsParamInst name="maxclient" key="maxclient" value="0" />
        <vnsParamInst name="maxreq" key="maxreq" value="0" />
        <vnsParamInst name="tcpb" key="tcpb" value="NO" />
        <vnsCfgRelInst name="MonitorConfig" key="MonitorConfig" targetName="monitor1"/>
    </vnsFolderInst>

    <vnsFolderInst ctrctNameOrLbl="any" graphNameOrLbl="G2" nodeNameOrLbl="any" key="Network">
      name="Network">
        <vnsFolderInst ctrctNameOrLbl="any" graphNameOrLbl="G2" nodeNameOrLbl="any" key="vip">
          name="vip">
            <vnsParamInst name="vipaddress1" key="vipaddress" value="10.10.10.200"/>
        </vnsFolderInst>
        <vnsFolderInst ctrctNameOrLbl="any" graphNameOrLbl="G2" nodeNameOrLbl="any" devCtxLbl="C1" key="snip" name="snip1">
          <vnsParamInst name="snipaddress" key="snipaddress" value="192.168.1.200"/>
        </vnsFolderInst>
    </vnsFolderInst>
  </vnsFolderInst>
</fvAEPg>
```

```

</vnsFolderInst>
<vnsFolderInst ctrctNameOrLbl="any" graphNameOrLbl="G2" nodeNameOrLbl="any"
    devCtxLbl="C2" key="snip" name="snip2">
<vnsFolderInst ctrctNameOrLbl="any" graphNameOrLbl="G1" nodeNameOrLbl="any" key="Network"
    name="Network">
    <vnsFolderInst ctrctNameOrLbl="any" graphNameOrLbl="G1" nodeNameOrLbl="any" key="vip"
        name="vip">
            <vnsParamInst name="vipaddress1" key="vipaddress" value="10.10.10.100"/>
        </vnsFolderInst>
        <vnsFolderInst ctrctNameOrLbl="any" graphNameOrLbl="G1" nodeNameOrLbl="any"
            devCtxLbl="C1" key="snip" name="snip1">
                <vnsParamInst name="snipaddress" key="snipaddress" value="192.168.1.100"/>
            </vnsFolderInst>
            <vnsFolderInst ctrctNameOrLbl="any" graphNameOrLbl="G1" nodeNameOrLbl="any"
                devCtxLbl="C2" key="snip" name="snip2">
                    <vnsParamInst name="snipaddress" key="snipaddress" value="192.168.1.101"/>
                </vnsFolderInst>
                <vnsFolderInst ctrctNameOrLbl="any" graphNameOrLbl="G1" nodeNameOrLbl="any"
                    devCtxLbl="C3" key="snip" name="snip3">
                        <vnsParamInst name="snipaddress" key="snipaddress" value="192.168.1.102"/>
                    </vnsFolderInst>
                </vnsFolderInst>
            </vnsFolderInst>
        <!-- SLB Configuration -->
        <vnsFolderInst ctrctNameOrLbl="any" graphNameOrLbl="any" nodeNameOrLbl="any" key="VServer"
            name="VServer">
            <!-- Virtual Server Configuration -->
            <vnsParamInst name="port" key="port" value="8010"/>
            <vnsParamInst name="vip" key="vip" value="10.10.10.100"/>
            <vnsParamInst name="vservername" key="vservername" value="crpvgrtst02-vip-8010"/>
            <vnsParamInst name="servicename" key="servicename" value="crpvgrtst02-8010"/>
            <vnsFolderInst ctrctNameOrLbl="any" graphNameOrLbl="any" nodeNameOrLbl="any"
                key="VServerGlobalConfig" name="VServerGlobalConfig">
                <vnsCfgRelInst name="ServiceConfig" key="ServiceConfig" targetName="Service1"/>
                <vnsCfgRelInst name="VipConfig" key="VipConfig" targetName="Network/vip"/>
            </vnsFolderInst>
        </vnsFolderInst>
    </vnsFolderInst>
</fvAEPg>

```

Example XML of Configuration Parameters Inside the Device Package

The following XML example shows configuration parameters inside of the device package:

```

<vnsMFolder key="VServer" scopedBy="epg">
    <vnsRsConnector tDn="uni/infra/mDev-Acme-ADC-1.0/mFunc-SLB/mConn-external"/>
    <vnsMParam key="vservername" description="Name of VServer" mandatory="true"/>
    <vnsMParam key="vip" description="Virtual IP"/>
    <vnsMParam key="subnet" description="Subnet IP"/>
    <vnsMParam key="port" description="Port for Virtual server"/>
    <vnsMParam key="persistencetype" description="persistencetype"/>
    <vnsMParam key="servicename" description="Service bound to this vServer"/>
    <vnsMParam key="servicetype" description="Service bound to this vServer"/>
    <vnsMParam key="clttimeout" description="Client timeout"/>
    <vnsMFolder key="VServerGlobalConfig"
        description="This references the global configuration">
        <vnsMRel key="ServiceConfig">
            <vnsRsTarget tDn="uni/infra/mDev-Acme-ADC-1.0/mDevCfg/mFolder-Service"/>
        </vnsMRel>
        <vnsMRel key="ServerConfig">
            <vnsRsTarget tDn="uni/infra/mDev-Acme-ADC-1.0/mDevCfg/mFolder-Server"/>
        </vnsMRel>
        <vnsMRel key="VipConfig">
            <vnsRsTarget
                tDn="uni/infra/mDev-Acme-ADC-1.0/mDevCfg/mFolder-Network/mFolder-vip"/>
            <vnsRsConnector tDn="uni/infra/mDev-Acme-ADC-1.0/mFunc-SLB/mConn-external"/>
        </vnsMRel>
    </vnsMFolder>

```

```
</vnsMFolder>
</vnsMFolder>
```

Example XML POST for an Abstract Function Node With Configuration Parameters

The following XML POST example shows an abstract function node with configuration parameters:

```
<vnsAbsNode name = "SLB" funcType="GoTo" >
    <vnsRsDefaultScopeToTerm tDn="uni/tn-tenant1/AbsGraph-G3/AbsTermNode-Output1/outtmnl"/>

    <vnsAbsFuncConn name = "C4" direction = "input">
        <vnsRsMConnAtt tDn="uni/infra/mDev-Acme-ADC-1.0/mFunc-SLB/mConn-external" />
    </vnsAbsFuncConn>
    <vnsAbsFuncConn name = "C5" direction = "output">
        <vnsRsMConnAtt tDn="uni/infra/mDev-Acme-ADC-1.0/mFunc-SLB/mConn-internal" />
    </vnsAbsFuncConn>

    <vnsAbsDevCfg>
        <vnsAbsFolder key="Network" name="Network" scopedBy="epg">
            <!-- Following scopes this folder to input terminal or Src Epg -->
            <vnsRsScopeToTerm tDn="uni/tn-tenant1/AbsGraph-G3/AbsTermNode-Output1/outtmnl"/>

            <!-- VIP address -->
            <vnsAbsFolder key="vip" name="vip" scopedBy="epg">
                <vnsAbsParam name="vipaddress" key="vipaddress" value="" />
            </vnsAbsFolder>

            <!-- SNIP address -->
            <vnsAbsFolder key="snip" name="snip" scopedBy="epg">
                <vnsAbsParam name="snipaddress" key="snipaddress" value="" />
            </vnsAbsFolder>
        </vnsAbsFolder>

        <vnsAbsFolder key="Service" name="Service" scopedBy="epg" cardinality="n">
            <vnsRsScopeToTerm tDn="uni/tn-tenant1/AbsGraph-G3/AbsTermNode-Output1/outtmnl"/>

            <vnsAbsParam name="servicename" key="servicename" value="" />
            <vnsAbsParam name="servername" key="servername" value="" />
            <vnsAbsParam name="serveripaddress" key="serveripaddress" value="" />
        </vnsAbsFolder>
    </vnsAbsDevCfg>

    <vnsAbsFuncCfg>
        <vnsAbsFolder key="VServer" name="VServer" scopedBy="epg">
            <vnsRsScopeToTerm tDn="uni/tn-tenant1/AbsGraph-G3/AbsTermNode-Output1/outtmnl"/>

            <!-- Virtual Server Configuration -->
            <vnsAbsParam name="vip" key="vip" value="" />
            <vnsAbsParam name="vservername" key="vservername" value="" />
            <vnsAbsParam name="servicename" key="servicename" />
            <vnsRsCfgToConn tDn="uni/tn-tenant1/AbsGraph-G3/AbsNode-Node2/AbsFConn-C4" />
        </vnsAbsFolder>
    </vnsAbsFuncCfg>
    <vnsRsNodeToMFunc tDn="uni/infra/mDev-Acme-ADC-1.0/mFunc-SLB" />
</vnsAbsNode>
```

Example XML POST for an Abstract Function Profile With Configuration Parameters

The following XML POST example shows an abstract function profile with configuration parameters:

```
<vnsAbsFuncProfContr name = "NP">
    <vnsAbsFuncProfGrp name = "Grp1">
        <vnsAbsFuncProf name = "P1">
            <vnsRsProfToMFunc tDn="uni/infra/mDev-Acme-ADC-1.0/mFunc-SLB" />
            <vnsAbsDevCfg name="D1">
                <vnsAbsFolder key="Service" name="Service-Default" cardinality="n">
                    <vnsAbsParam name="servicetype" key="servicetype" value="TCP" />
                </vnsAbsFolder>
            </vnsAbsDevCfg>
        </vnsAbsFuncProf>
    </vnsAbsFuncProfGrp>
</vnsAbsFuncProfContr>
```

```

<vnsAbsParam name="serviceport" key="serviceport" value="80"/>
<vnsAbsParam name="maxclient" key="maxclient" value="1000"/>
<vnsAbsParam name="maxreq" key="maxreq" value="100"/>
<vnsAbsParam name="cip" key="cip" value="enable"/>
<vnsAbsParam name="usip" key="usip" value="enable"/>
<vnsAbsParam name="sp" key="sp" value=""/>
<vnsAbsParam name="svrtimeout" key="svrtimeout" value="60"/>
<vnsAbsParam name="clttimeout" key="clttimeout" value="60"/>
<vnsAbsParam name="cka" key="cka" value="NO"/>
<vnsAbsParam name="tcpb" key="tcpb" value="NO"/>
<vnsAbsParam name="cmp" key="cmp" value="NO"/>
</vnsAbsFolder>
</vnsAbsDevCfg>
<vnsAbsFuncCfg name="SLB">
  <vnsAbsFolder key="VServer" name="VServer-Default">
    <vnsAbsParam name="port" key="port" value="80"/>
    <vnsAbsParam name="persistencetype" key="persistencetype"
      value="cookie"/>
    <vnsAbsParam name="clttimeout" key="clttimeout" value="100"/>
    <vnsAbsParam name="servicetype" key="servicetype" value="TCP"/>
    <vnsAbsParam name="servicename" key="servicename"/>
  </vnsAbsFolder>
  </vnsAbsFuncCfg>
</vnsAbsFuncPrcf>
</vnsAbsFuncProfGrp>
</vnsAbsFuncProfContr>

```

Copy Services

About Copy Services

Unlike Switched Port Analyzer (SPAN), which duplicates all traffic, the Cisco Application Centric Infrastructure (ACI) copy services feature enables selectively copying portions of the traffic between endpoint groups, according to the specifications of the contract. Broadcast, unknown unicast and multicast (BUM), and control plan traffic not covered by the contract are not copied. In contrast, SPAN copies everything out of endpoint groups, access ports, or uplink ports. Unlike SPAN, copy services do not add headers to the copied traffic. Copy service traffic is managed internally in the switch to minimize impact on normal traffic forwarding.

For more information about deploying Layer 4 to Layer 7 services, see the *Cisco APIC Layer 4 to Layer 7 Services Deployment Guide*.

Configuring Copy Services Using the REST API

A copy device is used as part of the copy services feature to create a copy node. A copy node specifies at which point of the data flow between endpoint groups to copy traffic.

This procedure provides examples of using the REST API to configure copy services.



Note

When you configure a copy device, the context aware parameter is not used. The context aware parameter has a default value of `single context`, which can be ignored.

Before You Begin

You must have configured a tenant.

Procedure

Step 1 Create a copy device.

Example:

```
<vnsLDevVip contextAware="single-Context" devtype="PHYSICAL" funcType="None" isCopy="yes"
managed="no"
    mode="legacy-Mode" name="copy0" packageModel="" svcType="COPY" trunking="no">
        <vnsRsALDevToPhysDomP tDn="uni/phys-phys_scale_copy"/>
        <vnsCDev devCtxLbl="" name="copy_Dyn_Device_0" vcenterName="" vmName="">
            <vnsCIf name="int1" vnicName="">
                <vnsRsCIfPathAtt tDn="topology/pod-1/paths-104/pathep-[eth1/15]"/>
            </vnsCIf>
            <vnsCIf name="int2" vnicName="">
                <vnsRsCIfPathAtt tDn="topology/pod-1/paths-105/pathep-[eth1/15]"/>
            </vnsCIf>
        </vnsCDev>
        <vnsLIf encap="vlan-3540" name="TAP">
            <vnsRsCIfAttN tDn="uni/tn-t22/1DevVip-copy0/cDev-copy_Dyn_Device_0/cIf-[int2]"/>
            <vnsRsCIfAttN tDn="uni/tn-t22/1DevVip-copy0/cDev-copy_Dyn_Device_0/cIf-[int1]"/>
        </vnsLIf>
    </vnsLDevVip>
```

Step 2 Create a logical device context (also known as a device selection policy).

Example:

```
<vnsLDevCtx ctrctNameOrLbl="c0" descr="" graphNameOrLbl="g0" name="" nodeNameOrLbl="CP1">
    <vnsRsLDevCtxToLDev tDn="uni/tn-t22/1DevVip-copy0"/>
    <vnsLIfCtx connNameOrLbl="copy" descr="" name="">
        <vnsRsLIfCtxToLIf tDn="uni/tn-t22/1DevVip-copy0/lIf-TAP"/>
    </vnsLIfCtx>
</vnsLDevCtx>
```

Step 3 Create and apply the copy graph template.

Example:

```
<vnsAbsGraph descr="" name="g0" ownerKey="" ownerTag="" uiTemplateType="UNSPECIFIED">
    <vnsAbsTermNodeCon descr="" name="T1" ownerKey="" ownerTag="">
        <vnsAbsTermConn attNotify="no" descr="" name="1" ownerKey="" ownerTag="">
            <vnsInTerm descr="" name="" />
            <vnsOutTerm descr="" name="" />
        </vnsAbsTermConn>
    <vnsAbsTermNodeProv descr="" name="T2" ownerKey="" ownerTag="">
        <vnsAbsTermConn attNotify="no" descr="" name="1" ownerKey="" ownerTag="">
            <vnsInTerm descr="" name="" />
            <vnsOutTerm descr="" name="" />
        </vnsAbsTermConn>
    </vnsAbsTermNodeProv>
    <vnsAbsConnection adjType="L2" connDir="provider" connType="external" descr="" name="C1"
        ownerKey="" ownerTag="" unicastRoute="yes">
        <vnsRsAbsConnectionConns tDn="uni/tn-t22/AbsGraph-g0/AbsTermNodeCon-T1/AbsTConn"/>
        <vnsRsAbsConnectionConns tDn="uni/tn-t22/AbsGraph-g0/AbsTermNodeProv-T2/AbsTConn"/>
        <vnsRsAbsCopyConnection tDn="uni/tn-t22/AbsGraph-g0/AbsNode-CP1/AbsFConn-copy"/>
    </vnsAbsConnection>
    <vnsAbsNode descr="" funcTemplateType="OTHER" funcType="None" isCopy="yes" managed="no"
        name="CP1" ownerKey="" ownerTag="" routingMode="unspecified" sequenceNumber="0"
        shareEncap="no">
        <vnsAbsFuncConn attNotify="no" descr="" name="copy" ownerKey="" ownerTag="">
            <vnsRsNodeToLDev tDn="uni/tn-t22/1DevVip-copy0"/>
        </vnsAbsFuncConn>
    </vnsAbsNode>
</vnsAbsGraph>
```

Step 4 Define the relation to the copy graph in the contract that is associated with the endpoint groups.

Example:

```
<vzBrCP descr="" name="c0" ownerKey="" ownerTag="" prio="unspecified" scope="tenant"
targetDscp="unspecified">
    <vzSubj consMatchT="AtleastOne" descr="" name="Subject" prio="unspecified"
provMatchT="AtleastOne"
        revFltPorts="yes" targetDscp="unspecified">
            <vzRsSubjFiltAtt directives="" tnVzFilterName="default"/>
                <vzRsSubjGraphAtt directives="" tnVnsAbsGraphName="g0"/>
            </vzSubj>
        </vzBrCP>
```

- Step 5** Attach the contract to the endpoint group.

Example:

```
<fvAEPg name="epg2860">
    <fvRsCons tnVzBrCPName="c0"/>
    <fvRsBd tnFvBDName="bd0"/>
    <fvRsDomAtt tDn="uni/phys-phys_scale_SB"/>
    <fvRsPathAtt tDn="topology/pod-1/paths-104/pathep-[PC_int2_g1]" encap="vlan-2860"
        instrImedcy="immediate"/>
</fvAEPg>
<fvAEPg name="epg2861">
    <fvRsProv tnVzBrCPName="c0"/>
    <fvRsBd tnFvBDName="bd0"/>
    <fvRsDomAtt tDn="uni/phys-phys_scale_SB"/>
    <fvRsPathAtt tDn="topology/pod-1/paths-105/pathep-[PC_policy]" encap="vlan-2861"
        instrImedcy="immediate"/>
</fvAEPg>
```

Developing Automation

About the REST APIs

Automation relies on the Application Policy Infrastructure Controller (APIC) northbound Representational State Transfer (REST) APIs. Anything that can be done through the APIC UI can also be done using XML-based REST POSTs using the northbound APIs. For example, you can monitor events through those APIs, dynamically enable EPGs, and add policies.

You can also use the northbound REST APIs to monitor for notifications that a device has been brought onboard, and to monitor faults. In both cases, you can monitor events that trigger specific actions. For example, if you see faults that occur on a specific application tier and determine that there is a loss of connectivity and a leaf node is going down, you can trigger an action to redeploy those applications somewhere else. If you have certain contracts on which you detect packet drops occurring, you could enable some copies of those contracts on the particular application. You can also use a statistics monitoring policy, where you monitor certain counters because of issues that have been reported.

For information on how to construct the XML files submitted to the APIC northbound API, see *Cisco APIC Layer 4 to Layer 7 Device Package Development Guide*.

The following Python APIs, defined in the *Cisco APIC Management Information Model Reference* can be used to submit REST POST calls using the northbound API:

- `vns:LDevVip`: Upload a device cluster
- `vns:CDev`: Upload a device

- vns:LIf: Create logical interfaces
- vns:AbsGraph: Create a graph
- vz:BrCP: Attach a graph to a contract

Examples of Automating Using the REST APIs

This section contains examples of using the REST APIs to automate tasks.

The following REST request creates a tenant with a broadcast domain, a Layer 3 network, application endpoint groups, and an application profile:

```
<polUni>
  <fvTenant dn="uni/tn-acme" name="acme">

    <!--L3 Network-->
    <fvCtx name="MyNetwork"/>

    <!-- Bridge Domain for MySrvr EPG -->
    <fvBD name="MySrvrBD">
      <fvRsCtx tnFvCtxName="MyNetwork"/>
      <fvSubnet ip="10.10.10.10/24">
        </fvSubnet>
    </fvBD>

    <!-- Bridge Domain for MyClnt EPG -->
    <fvBD name="MyClntBD">
      <fvRsCtx tnFvCtxName="MyNetwork"/>
      <fvSubnet ip="20.20.20.20/24">
        </fvSubnet>
    </fvBD>

    <fvAp dn="uni/tn-acme/ap-MyAP" name="MyAP">

      <fvAEPg dn="uni/tn-acme/ap-MyAP/epg-MyClnt" name="MyClnt">
        <fvRsBd tnFvBDName="MySrvrBD"/>
        <fvRsDomAtt tDn="uni/vmmp-Vendor1/dom-MyVMs"/>
        <fvRsProv tnVzBrCName="webCtrct" />
        <fvRsPathAtt tDn="topology/pod-1/paths-17/pathep-[eth1/21]" encaps="vlan-202"/>
        <fvRsPathAtt tDn="topology/pod-1/paths-18/pathep-[eth1/21]" encaps="vlan-202"/>
      </fvAEPg>

      <fvAEPg dn="uni/tn-acme/ap-MyAP/epg-MySRVR" name="MySRVR">
        <fvRsBd tnFvBDName="MyClntBD"/>
        <fvRsDomAtt tDn="uni/vmmp-Vendor1/dom-MyVMs"/>
        <fvRsCons tnVzBrCName="webCtrct" />
        <fvRsPathAtt tDn="topology/pod-1/paths-17/pathep-[eth1/21]" encaps="vlan-203"/>
        <fvRsPathAtt tDn="topology/pod-1/paths-18/pathep-[eth1/21]" encaps="vlan-203"/>
      </fvAEPg>
    </fvAp>
  </fvTenant>
</polUni>
```

The following REST request creates a VLAN namespace:

```
<polUni>
  <infraInfra>
    <fvnsVlanInstP name="MyNS" allocMode="dynamic">
      <fvnsEncapBlk name="encap" from="vlan-201" to="vlan-300"/>
    </fvnsVlanInstP>
  </infraInfra>
</polUni>
```

The following REST request creates a VMM domain:

```
<polUni>
  <vmmProvP vendor="Vendor1">
    <vmmDomP name="MyVMs">
      <infraRsVlanNs tDn="uni/infra/vlanns-MyNS-dynamic"/>
      <vmmUsrAccP name="admin" usr="administrator" pwd="in$1eme"/>
      <vmmCtrlrP name="vcenter1" hostOrIp="192.168.64.186">
        <vmmRsAcc tDn="uni/vmmp-Vendor1/dom-MyVMs/usracc-admin"/>
      </vmmCtrlrP>
    </vmmDomP>
  </vmmProvP>
</polUni>
```

The following REST request creates a physical domain:

```
<polUni>
  <physDomP name="phys">
    <infraRsVlanNs tDn="uni/infra/vlanns-MyNS-dynamic"/>
  </physDomP>
</polUni>
```

The following REST request creates a managed device cluster:

```
<polUni>
  <fvTenant dn="uni/tn-acme" name="acme">
    <vnsLDevVip name="ADCCluster1" contextAware=1>
      <vnsRsMDevAtt tDn="uni/infra/mDev-Acme-ADC-1.0"/>
      <vnsRsDevEpg tDn="uni/tn-acme/ap-services/epg-ifc"/>
      <vnsRsALDevToPhysDomP tDn="uni/phys-phys"/>

      <vnsCMgmt name="devMgmt" host="42.42.42.100" port="80"/>
      <vnsCCred name="username" value="admin"/>

      <vnsCCredSecret name="password" value="admin"/>
    </vnsLDevVip>
  </fvTenant>
</polUni>
```

The following REST request creates an unmanaged device cluster:

```
<polUni>
  <fvTenant name="HA_Tenant1">
    <vnsLDevVip name="ADCCluster1" devtype="VIRTUAL" managed="no">
      <vnsRsALDevToDomP tDn="uni/vmmp-VMware/dom-mininet"/>
    </vnsLDevVip>
  </fvTenant>
</polUni>
```

The following REST request creates a device cluster context:

```
<polUni>
  <fvTenant dn="uni/tn-acme" name="acme">
    <vnsLDevCtx ctrctNameOrLbl="webCtrct" graphNameOrLbl="G1" nodeNameOrLbl="Node1">
      <vnsRsLDevCtxToLDev tDn="uni/tn-acme/lDevVip-ADCCluster1"/>
      <vnsLIfCtx connNameOrLbl="ssl-inside">
        <vnsRsLIfCtxToLIf tDn="uni/tn-acme/lDevVip-ADCCluster1/lIf-int"/>
      </vnsLIfCtx>
      <vnsLIfCtx connNameOrLbl="any">
        <vnsRsLIfCtxToLIf tDn="uni/tn-acme/lDevVip-ADCCluster1/lIf-ext"/>
      </vnsLIfCtx>
    </vnsLDevCtx>
  </fvTenant>
</polUni>
```

The following REST request creates a device cluster context used in route peering:

```
<polUni>
  <fvTenant dn="uni/tn-coke{{tenantId}}" name="coke{{tenantId}}">
    <vnsRtrCfg name="Dev1Ctx1" rtrId="180.0.0.12"/>
    <vnsLDevCtx ctrctNameOrLbl="webCtrct1" graphNameOrLbl="WebGraph">
      <nodeNameOrLbl="FW">
        <vnsRsLDevCtxToLDev tDn="uni/tn-tenant1/lDevVip-Firewall"/>
      </nodeNameOrLbl>
    </vnsLDevCtx>
  </fvTenant>
</polUni>
```

```

<vnsRsLDevCtxToRtrCfg tnVnsRtrCfgName="FwRtrCfg"/>
<vnsLIfCtx connNameOrLbl="internal">
    <vnsRsLIfCtxToInstP tDn="uni/tn-tenant1/out-OspfInternal/instP-IntInstP"
        status="created,modified"/>
    <vnsRsLIfCtxToLIf tDn="uni/tn-tenant1/lDevVip-Firewall/lIf-internal"/>
</vnsLIfCtx>
<vnsLIfCtx connNameOrLbl="external">
    <vnsRsLIfCtxToInstP tDn="uni/tn-common/out-OspfExternal/instP-ExtInstP"
        status="created,modified"/>
    <vnsRsLIfCtxToLIf tDn="uni/tn-tenant1/lDevVip-Firewall/lIf-external"/>
</vnsLIfCtx>
</vnsLDevCtx>
</fvTenant>
</polUni>

```



Note For information about configuring external connectivity for tenants (a Layer 3 outside), see the *Cisco APIC Basic Configuration Guide*.

The following REST request adds a logical interface in a device cluster:

```

<polUni>
    <fvTenant dn="uni/tn-acme" name="acme">
        <vnsLDevVip name="ADCCluster1">
            <vnsLIf name="C5">
                <vnsRsMetaIf tDn="uni/infra/mDev-Acme-ADC-1.0/mIfLbl-outside"/>
                <vnsRsCIfAtt tDn="uni/tn-acme/lDevVip-ADCCluster1/cDev-ADC1/cIf-int"/>
            </vnsLIf>
            <vnsLIf name="C4">
                <vnsRsMetaIf tDn="uni/infra/mDev-Acme-ADC-1.0/mIfLbl-inside"/>
                <vnsRsCIfAtt tDn="uni/tn-acme/lDevVip-ADCCluster1/cDev-ADC1/cIf-ext"/>
            </vnsLIf>
        </vnsLDevVip>
    </fvTenant>
</polUni>

```

The following REST request adds a concrete device in a physical device cluster:

```

<polUni>
    <fvTenant dn="uni/tn-acme" name="acme">
        <vnsLDevVip name="ADCCluster1">
            <vnsCDev name="ADC1" devCtxLbl="C1">
                <vnsCIf name="int">
                    <vnsRsCIfPathAtt tDn="topology/pod-1/paths-17/pathep-[eth1/22]"/>
                </vnsCIf>
                <vnsCIf name="ext">
                    <vnsRsCIfPathAtt tDn="topology/pod-1/paths-17/pathep-[eth1/21]"/>
                </vnsCIf>
                <vnsCIf name="mgmt">
                    <vnsRsCIfPathAtt tDn="topology/pod-1/paths-17/pathep-[eth1/20]"/>
                </vnsCIf>
                <vnsCMgmt name="devMgmt" host="172.30.30.100" port="80"/>
                <vnsCCred name="username" value="admin"/>
                <vnsCCred name="password" value="admin"/>
            </vnsCDev>
            <vnsCDev name="ADC2" devCtxLbl="C2">
                <vnsCIf name="int">
                    <vnsRsCIfPathAtt tDn="topology/pod-1/paths-17/pathep-[eth1/23]"/>
                </vnsCIf>
                <vnsCIf name="ext">
                    <vnsRsCIfPathAtt tDn="topology/pod-1/paths-17/pathep-[eth1/24]"/>
                </vnsCIf>
                <vnsCIf name="mgmt">
                    <vnsRsCIfPathAtt tDn="topology/pod-1/paths-17/pathep-[eth1/30]"/>
                </vnsCIf>
                <vnsCMgmt name="devMgmt" host="172.30.30.200" port="80"/>
                <vnsCCred name="username" value="admin"/>
                <vnsCCred name="password" value="admin"/>
            </vnsCDev>
        </vnsLDevVip>

```

```

        </fvTenant>
</polUni>
```

The following REST request adds a concrete device in a virtual device cluster:

```

<polUni>
    <fvTenant dn="uni/tn-coke5" name="coke5">
        <vnsLDevVip name="Firewall15" devtype="VIRTUAL">
            <vnsCDev name="ASA5" vcenterName="vcenter1" vmName="ifav16-ASAv-scale-05">
                <vnsCIf name="Gig0/0" vnicName="Network adapter 2"/>
                <vnsCIf name="Gig0/1" vnicName="Network adapter 3"/>
                <vnsCIf name="Gig0/2" vnicName="Network adapter 4"/>
                <vnsCIf name="Gig0/3" vnicName="Network adapter 5"/>
                <vnsCIf name="Gig0/4" vnicName="Network adapter 6"/>
                <vnsCIf name="Gig0/5" vnicName="Network adapter 7"/>
                <vnsCIf name="Gig0/6" vnicName="Network adapter 8"/>
                <vnsCIf name="Gig0/7" vnicName="Network adapter 9"/>
                <vnsCMgmt name="devMgmt" host="3.5.3.170" port="443"/>
                <vnsCCred name="username" value="admin"/>
                <vnsCCredSecret name="password" value="insieme"/>
            </vnsCDev>
        </vnsLDevVip>
    </fvTenant>
</polUni>
```

The following REST request creates a service graph in managed mode:

```

<polUni>
    <fvTenant name="acme">
        <vnsAbsGraph name = "G1">

        <vnsAbsTermNode name = "Input1">
            <vnsAbsTermConn name = "C1" direction = "output">
            </vnsAbsTermConn>
        </vnsAbsTermNode>

        <!-- Node1 Provides SLB functionality -->
        <vnsAbsNode name = "Node1" funcType="GoTo" >
            <vnsRsDefaultScopeToTerm
                tDn="uni/tn-acme/AbsGraph-G1/AbsTermNode-Output1/outtmnl"/>

            <vnsAbsFuncConn name = "C4" direction = "input">
                <vnsRsMConnAtt tDn="uni/infra/mDev-Acme-ADC-1.0/mFunc-SLB/mConn-external"/>

                <vnsRsConnToLIf tDn="uni/tn-acme/lDevVip-ADCCluster1/lIf-C4"/>
            </vnsAbsFuncConn>

            <vnsAbsFuncConn name = "C5" direction = "output">
                <vnsRsMConnAtt tDn="uni/infra/mDev-Acme-ADC-1.0/mFunc-SLB/mConn-internal"/>

                <vnsRsConnToLIf tDn="uni/tn-acme/lDevVip-ADCCluster1/lIf-C5"/>
            </vnsAbsFuncConn>

            <vnsRsNodeToMFunc tDn="uni/infra/mDev-Acme-ADC-1.0/mFunc-SLB"/>
        </vnsAbsNode>

        <vnsAbsTermNode name = "Output1">
            <vnsAbsTermConn name = "C6" direction = "input">
            </vnsAbsTermConn>
        </vnsAbsTermNode>

        <vnsAbsConnection name = "CON1">
            <vnsRsAbsConnectionConns
                tDn="uni/tn-acme/AbsGraph-G1/AbsTermNode-Input1/AbsTConn"/>
            <vnsRsAbsConnectionConns
                tDn="uni/tn-acme/AbsGraph-G1/AbsNode-Node1/AbsFConn-C4"/>
        </vnsAbsConnection>

        <vnsAbsConnection name = "CON3">
            <vnsRsAbsConnectionConns
                tDn="uni/tn-acme/AbsGraph-G1/AbsNode-Node1/AbsFConn-C5"/>
            <vnsRsAbsConnectionConns
                tDn="uni/tn-acme/AbsGraph-G1/AbsTermNode-Output1/AbsTConn"/>
        </vnsAbsConnection>
```

```

    </vnsAbsGraph>
    </fvTenant>
</polUni>
```

The following REST request creates a service graph in unmanaged mode:

```

<polUni>
    <fvTenant name="HA_Tenant1">
        <vnsAbsGraph name="g1">

            <vnsAbsTermNodeProv name="Input1">
                <vnsAbsTermConn name="C1">
                </vnsAbsTermConn>
            </vnsAbsTermNodeProv>

            <!-- Node1 Provides LoadBalancing functionality -->
            <vnsAbsNode name="Node1" managed="no">
                <vnsRsDefaultScopeToTerm
                    tDn="uni/tn-HA_Tenant1/AbsGraph-g1/AbsTermNodeProv-Input1/outtmnl"/>
                <vnsAbsFuncConn name="outside" attNotify="true">
                </vnsAbsFuncConn>
                <vnsAbsFuncConn name="inside" attNotify="true">
                </vnsAbsFuncConn>
            </vnsAbsNode>

            <vnsAbsTermNodeCon name="Output1">
                <vnsAbsTermConn name="C6">
                </vnsAbsTermConn>
            </vnsAbsTermNodeCon>

            <vnsAbsConnection name="CON2" adjType="L3" unicastRoute="yes">
                <vnsRsAbsConnectionConns
                    tDn="uni/tn-HA_Tenant1/AbsGraph-g1/AbsTermNodeCon-Output1/AbsTConn"/>
                <vnsRsAbsConnectionConns
                    tDn="uni/tn-HA_Tenant1/AbsGraph-g1/AbsNode-Node1/AbsFConn-outside"/>
            </vnsAbsConnection>

            <vnsAbsConnection name="CON1" adjType="L2" unicastRoute="no">
                <vnsRsAbsConnectionConns
                    tDn="uni/tn-HA_Tenant1/AbsGraph-g1/AbsNode-Node1/AbsFConn-inside"/>
                <vnsRsAbsConnectionConns
                    tDn="uni/tn-HA_Tenant1/AbsGraph-g1/AbsTermNodeProv-Input1/AbsTConn"/>
            </vnsAbsConnection>

        </vnsAbsGraph>
    </fvTenant>
</polUni>
```

The following REST request creates a security policy (contract):

```

<polUni>
    <fvTenant dn="uni/tn-acme" name="acme">
        <vzFilter name="HttpIn">
            <vzEntry name="e1" prot="6" dToPort="80"/>
        </vzFilter>

        <vzBrCP name="webCtrct">
            <vzSubj name="http">
                <vzRsSubjFiltAtt tnVzFilterName="HttpIn"/>
            </vzSubj>
        </vzBrCP>
    </fvTenant>
</polUni>
```

The following REST request provides graph configuration parameters from an application EPG:

```

<polUni>
    <fvTenant dn="uni/tn-acme" name="acme">

        <!-- Application Profile -->
        <fvAp dn="uni/tn-acme/ap-MyAP" name="MyAP">

            <!-- EPG 1 -->
            <fvAEpg dn="uni/tn-acme/ap-MyAP/epg-MyClnt" name="MyClnt">
```

```

<fvRsBd tnFvBDName="MyClntBD"/>
<fvRsDomAtt tDn="uni/vmmp-Vendor1/dom-MyVMs"/>
<fvRsProv tnVzBrCPName="webCtrct">
</fvRsProv>
<fvRsPathAtt tDn="topology/pod-1/paths-17/pathep-[eth1/20]" encap="vlan-201"/>
<fvSubnet name="SrcSubnet" ip="192.168.10.1/24"/>
</fvAEPg>

<!-- EPG 2 -->
<fvAEPg dn="uni/tn-acme/ap-MyAP/epg-MySRVR" name="MySRVR">
<fvRsBd tnFvBDName="MyClntBD"/>
<fvRsDomAtt tDn="uni/vmmp-Vendor1/dom-MyVMs"/>
<fvRsCons tnVzBrCPName="webCtrct">
</fvRsCons>

<vnsFolderInst ctrctNameOrLbl="any" graphNameOrLbl="any" nodeNameOrLbl="any"
key="Monitor" name="monitor1">
<vnsParamInst name="weight" key="weight" value="10"/>
</vnsFolderInst>

<vnsFolderInst ctrctNameOrLbl="any" graphNameOrLbl="any" nodeNameOrLbl="any"
key="Service" name="Service1">
<vnsParamInst name="servicename" key="servicename"
value="crpvgrst02-8010"/>
<vnsParamInst name="servicetype" key="servicetype" value="TCP"/>
<vnsParamInst name="servername" key="servername"
value="s192.168.100.100"/>
<vnsParamInst name="serveripaddress" key="serveripaddress"
value="192.168.100.100"/>
<vnsParamInst name="serviceport" key="serviceport" value="8080"/>
<vnsParamInst name="svrtimeout" key="svrtimeout" value="9000"/>
<vnsParamInst name="clttimeout" key="clttimeout" value="9000"/>
<vnsParamInst name="usip" key="usip" value="NO"/>
<vnsParamInst name="useproxyport" key="useproxyport" value="" />
<vnsParamInst name="cip" key="cip" value="ENABLED"/>
<vnsParamInst name="cka" key="cka" value="NO"/>
<vnsParamInst name="sp" key="sp" value="OFF"/>
<vnsParamInst name="cmp" key="cmp" value="NO"/>
<vnsParamInst name="maxclient" key="maxclient" value="0"/>
<vnsParamInst name="maxreq" key="maxreq" value="0"/>
<vnsParamInst name="tcpb" key="tcpb" value="NO"/>
<vnsCfgRelInst name="MonitorConfig" key="MonitorConfig"
targetName="monitor1"/>
</vnsFolderInst>

<vnsFolderInst ctrctNameOrLbl="any" graphNameOrLbl="any"
nodeNameOrLbl="any" key="Network" name="Network">
<vnsFolderInst ctrctNameOrLbl="any" graphNameOrLbl="any"
nodeNameOrLbl="any" key="vip" name="vip">
<vnsParamInst name="vipaddress1" key="vipaddress"
value="10.10.10.100"/>
</vnsFolderInst>
<vnsFolderInst ctrctNameOrLbl="any" graphNameOrLbl="any"
nodeNameOrLbl="any" devCtxLbl="C1" key="snip" name="snip1">
<vnsParamInst name="snipaddress" key="snipaddress"
value="192.168.1.100"/>
</vnsFolderInst>
<vnsFolderInst ctrctNameOrLbl="any" graphNameOrLbl="any"
nodeNameOrLbl="any" devCtxLbl="C2" key="snip" name="snip2">
<vnsParamInst name="snipaddress" key="snipaddress"
value="192.168.1.101"/>
</vnsFolderInst>
<vnsFolderInst ctrctNameOrLbl="any" graphNameOrLbl="any"
nodeNameOrLbl="any" devCtxLbl="C3" key="snip" name="snip3">
<vnsParamInst name="snipaddress" key="snipaddress"
value="192.168.1.102"/>
</vnsFolderInst>
</vnsFolderInst>

<!-- SLB Configuration -->
<vnsFolderInst ctrctNameOrLbl="any" graphNameOrLbl="any"
nodeNameOrLbl="any" key="VServer" name="VServer">

```

```
<!-- Virtual Server Configuration -->
<vnsParamInst name="port" key="port" value="8010"/>
<vnsParamInst name="vip" key="vip" value="10.10.10.100"/>
<vnsParamInst name="vservername" key="vservername"
    value="crpvgrtst02-vip-8010"/>
<vnsParamInst name="servicename" key="servicename"
    value="crpvgrtst02-8010"/>
<vnsParamInst name="servicetype" key="servicetype" value="TCP"/>
<vnsFolderInst ctrctNameOrLbl="any" graphNameOrLbl="any"
nodeNameOrLbl="any" key="VServerGlobalConfig" name="VServerGlobalConfig">

    <vnsCfgRelInst name="ServiceConfig" key="ServiceConfig"
        targetName="Service1"/>
    <vnsCfgRelInst name="VipConfig" key="VipConfig"
        targetName="Network/vip"/>
</vnsFolderInst>
</vnsFolderInst>
</fvAEPg>
</fvAp>
</fvTenant>
</polUni>
```

The following REST request attaches a service graph to a contract:

```
<polUni>
    <fvTenant name="acme">
        <vzBrCP name="webCtrct">
            <vzSubj name="http">
                <vzRsSubjGraphAtt graphName="G1" termNodeName="Input1"/>
            </vzSubj>
        </vzBrCP>
    </fvTenant>
</polUni>
```

Example: Configuring Layer 4 to Layer 7 Services (Firewall)

Example: Configuring Layer 4 to Layer 7 Services Using the REST API

This topic shows the steps for configuring Layer 4 to Layer 7 services (ASA Firewall) using the REST API.

Before You Begin

- Create the tenant to use the Layer 4 to Layer 7 services, with a Layer 3 outside network and bridge domains.
- Create application profiles.
- Configure a physical or VMM domain.
- Import and register the device packages and configure parameters for them.

Procedure

Step 1 Create a Layer 4 to Layer 7 **ASAv** device package model, using XML such as the following example:

Example:

```
<vnsLDevVip trunking="no" svcType="FW"
packageModel="ASAv" name="ASAv" mode="legacy-Mode"
managed="yes" isCopy="no" funcType="GoTo"
dn="uni/tn-Tenant-test/lDevVip-ASAv" devtype="VIRTUAL"
contextAware="single-Context">
```

```

<vnsCCred name="username" value="admin"/>
<vnsRsMDevAtt tDn="uni/infra/mDev-CISCO-ASA-1.2"/>
<vnsCCredSecret name="password"/>
<vnsCMgmt name="" port="443" host="172.31.184.249"/>
<vnsRsALDevToDomP tDn="uni/vmmp-VMware/dom-ACI_vDS"/>
<vnsCDev name="Device1" vmName="ASAv-L3" vcenterName="vcenter" devCtxLbl="">
<vnsCCred name="username" value="admin"/>
<vnsCCredSecret name="password"/>
<vnsCMgmt name="" port="443" host="172.31.184.249"/>
<vnsCIf name="GigabitEthernet0/1" vnicName="Network adapter 3"/>
<vnsCIf name="GigabitEthernet0/0" vnicName="Network adapter 2"/>
<vnsRsCDevToCtrlrP tDn="uni/vmmp-VMware/dom-ACI_vDS/ctrlr-vcenter"/>
</vnsCDev>

<vnsLIf name="provider" encaps="unknown">
<vnsRsMetaIf tDn="uni/infra/mDev-CISCO-ASA-1.2/mIfLbl-internal" isConAndProv="no"/>
<vnsRsCIfAttN tDn="uni/tn-Tenant-test/lDevVip-ASAv/cDev-Device1/cIf-[GigabitEthernet0/1]"/>
</vnsLIf>

<vnsLIf name="consumer" encaps="unknown">
<vnsRsMetaIf tDn="uni/infra/mDev-CISCO-ASA-1.2/mIfLbl-external" isConAndProv="no"/>
<vnsRsCIfAttN tDn="uni/tn-Tenant-test/lDevVip-ASAv/cDev-Device1/cIf-[GigabitEthernet0/0]"/>
</vnsLIf>
</vnsLDevVip>

```

Step 2 Configure a Layer 4 to Layer 7 **FW-Graph** using XML such as the following example:

Example:

```

<vnsAbsGraph uiTemplateType="UNSPECIFIED" ownerTag="" ownerKey="" name="FW-Graph"
dn="uni/tn-Tenant-test/AbsGraph-FW-Graph" descr="">

<vnsAbsTermNodeCon ownerTag="" ownerKey="" name="T1" descr="">
<vnsAbsTermConn ownerTag="" ownerKey="" name="1" descr="" attNotify="no"/>
<vnsInTerm name="" descr="" />
<vnsOutTerm name="" descr="" />
</vnsAbsTermNodeCon>

<vnsAbsTermNodeProv ownerTag="" ownerKey="" name="T2" descr="">
<vnsAbsTermConn ownerTag="" ownerKey="" name="1" descr="" attNotify="no"/>
<vnsInTerm name="" descr="" />
<vnsOutTerm name="" descr="" />
</vnsAbsTermNodeProv>

<vnsAbsConnection ownerTag="" ownerKey="" name="C1" descr="" unicastRoute="yes"
directConnect="no" connType="external" connDir="provider" adjType="L2">
<vnsRsAbsConnectionConns
tDn="uni/tn-Tenant-test/AbsGraph-FW-Graph/AbsNode-N1/AbsFConn-consumer"/>
<vnsRsAbsConnectionConns
tDn="uni/tn-Tenant-test/AbsGraph-FW-Graph/AbsTermNodeCon-T1/AbsTConn"/>
</vnsAbsConnection>

<vnsAbsConnection ownerTag="" ownerKey="" name="C2" descr="" unicastRoute="yes"
directConnect="no" connType="external" connDir="provider" adjType="L2">
<vnsRsAbsConnectionConns
tDn="uni/tn-Tenant-test/AbsGraph-FW-Graph/AbsNode-N1/AbsFConn-provider"/>
<vnsRsAbsConnectionConns
tDn="uni/tn-Tenant-test/AbsGraph-FW-Graph/AbsTermNodeProv-T2/AbsTConn"/>
</vnsAbsConnection>

<vnsAbsNode ownerTag="" ownerKey="" name="N1" descr="" shareEncap="no" sequenceNumber="0"
routingMode="unspecified" managed="yes" isCopy="no" funcType="GoTo"
funcTemplateType="FW_ROUTE">
<vnsAbsFuncConn ownerTag="" ownerKey="" name="consumer" descr="" attNotify="no">
<vnsRsMConnAtt tDn="uni/infra/mDev-CISCO-ASA-1.2/mFunc-Firewall/mConn-external"/>
</vnsAbsFuncConn>

<vnsAbsFuncConn ownerTag="" ownerKey="" name="provider" descr="" attNotify="no">
<vnsRsMConnAtt tDn="uni/infra/mDev-CISCO-ASA-1.2/mFunc-Firewall/mConn-internal"/>
</vnsAbsFuncConn>
<vnsRsNodeToAbsFuncProf
tDn="uni/infra/mDev-CISCO-ASA-1.2/absFuncProfContr/absFuncProfGrp-WebServiceProfileGroup/absFuncProf-WebPolicyForRoutedMode"/>

```

```
<vnsRsNodeToLDev tDn="uni/tn-Tenant-test/lDevVip-ASAv"/>
<vnsRsNodeToMFunc tDn="uni/infra/mDev-CISCO-ASA-1.2/mFunc-Firewall"/>
</vnsAbsNode>
</vnsAbsGraph>
```

Step 3 Create a device selection policy, using XML such as the following example:

Example:

```
<vnsLDevCtx nodeNameOrLbl="N1" name="" graphNameOrLbl="FW-Graph"
dn="uni/tn-Tenant-test/ldevCtx-c-Client-to-Web-g-FW-Graph-n-N1" descr=""
ctrctNameOrLbl="Client-to-Web">
<vnsRsLDevCtxToLDev tDn="uni/tn-Tenant-test/lDevVip-ASAv"/>

<vnsLIfCtx name="" descr="" connNameOrLbl="provider">
<vnsRsLIfCtxToBD tDn="uni/tn-Tenant-test/BD-BD2"/>
<vnsRsLIfCtxToLIf tDn="uni/tn-Tenant-test/lDevVip-ASAv/lIf-provider"/>
</vnsLIfCtx>

<vnsLIfCtx name="" descr="" connNameOrLbl="consumer">
<vnsRsLIfCtxToBD tDn="uni/tn-Tenant-test/BD-BD1"/>
<vnsRsLIfCtxToLIf tDn="uni/tn-Tenant-test/lDevVip-ASAv/lIf-consumer"/>
</vnsLIfCtx>
</vnsLDevCtx>
```

Step 4 Configure a contract, associated with the **FW-Graph** service graph template, using XML such as the following example:

Example:

```
<vzBrCP targetDscp="unspecified" scope="tenant" prio="unspecified" ownerTag=""
ownerKey="" name="Client-to-Web" dn="uni/tn-Tenant-test/brc-Client-to-Web" descr="">

<vzSubj targetDscp="unspecified" prio="unspecified" name="Subject" descr=""
revFltPorts="yes" provMatchT="AtleastOne" consMatchT="AtleastOne"
<vzRsSubjFiltAtt tnVzFilterName="default" directives="" />
<vzRsSubjGraphAtt directives="" tnVnsAbsGraphName="FW-Graph" />
</vzSubj>
</vzBrCP>
```

Step 5 Create the Client EPG, using XML such as the following example:

Example:

```
<fvAEPg prio="unspecified" pcEnfPref="unenforced" name="Client"
matchT="AtleastOne" isAttrBasedEPg="no" fwdCtrl="" dn="uni/tn-Tenant-test/ap-ANP/epg-Client"
descr="">
<fvRsCons prio="unspecified" tnVzBrCPName="Client-to-Web"/>
<fvRsDomAtt tDn="uni/vmmp-VMware/dom-ACI_vDS" resImedcy="lazy" primaryEncap="unknown"
instrImedcy="lazy" encap="unknown" delimiter="" classPref="encap"/>
<fvRsBd tnFvBDName="BD1"/>
<fvRsCustQosPol tnQosCustomPolName="" />
</fvAEPg>
```

Step 6 Create the Web EPG, using XML such as the following example:

Example:

```
<fvAEPg prio="unspecified" pcEnfPref="unenforced" name="Web" matchT="AtleastOne"
isAttrBasedEPg="no" fwdCtrl="" dn="uni/tn-Tenant-test/ap-ANP/epg-Web" descr="">
<fvRsDomAtt tDn="uni/vmmp-VMware/dom-ACI_vDS" resImedcy="lazy" primaryEncap="unknown"
instrImedcy="lazy" encap="unknown" delimiter="" classPref="encap"/>
<fvRsBd tnFvBDName="BD2"/>
<fvRsCustQosPol tnQosCustomPolName="" />

<vnsFolderInst name="internalIf" scopedBy="epg" nodeNameOrLbl="N1" locked="no" key="Interface"
graphNameOrLbl="FW-Graph" devCtxLbl="" ctrctNameOrLbl="Client-to-Web"
cardinality="unspecified">

<vnsFolderInst name="internalIfCfg" scopedBy="epg" nodeNameOrLbl="N1" locked="no"
key="InterfaceConfig"
```

```

graphNameOrLbl="FW-Graph" devCtxLbl="" ctrctNameOrLbl="Client-to-Web"
cardinality="unspecified">
<vnspParamInst name="internal_security_level" locked="no" key="security_level"
cardinality="unspecified"
value="100" validation="" mandatory="no"/>
</vnspFolderInst>
</vnspFolderInst>

<vnspFolderInst name="externalIf" scopedBy="epg" nodeNameOrLbl="N1" locked="no" key="Interface">
graphNameOrLbl="FW-Graph" devCtxLbl="" ctrctNameOrLbl="Client-to-Web"
cardinality="unspecified">
<vnspFolderInst name="ExtAccessGroup" scopedBy="epg" nodeNameOrLbl="N1" locked="no"
key="AccessGroup"
graphNameOrLbl="FW-Graph" devCtxLbl="" ctrctNameOrLbl="Client-to-Web"
cardinality="unspecified">
<vnspCfgrRelInst name="name" locked="no" key="inbound_access_list_name"
cardinality="unspecified" mandatory="no"
targetName="access-list-inbound"/>
</vnspFolderInst>

<vnspFolderInst name="externalIfCfg" scopedBy="epg" nodeNameOrLbl="N1" locked="no"
key="InterfaceConfig"
graphNameOrLbl="FW-Graph" devCtxLbl="" ctrctNameOrLbl="Client-to-Web"
cardinality="unspecified">
<vnspParamInst name="external_security_level" locked="no" key="security_level"
cardinality="unspecified" value="50" validation="" mandatory="no"/>
</vnspFolderInst>
</vnspFolderInst>

<vnspFolderInst name="IntConfig" scopedBy="epg" nodeNameOrLbl="N1" locked="no"
key="InIntfConfigRelFolder"
graphNameOrLbl="FW-Graph" devCtxLbl="" ctrctNameOrLbl="Client-to-Web"
cardinality="unspecified">
<vnspCfgrRelInst name="InConfigrel" locked="no" key="InIntfConfigRel"
cardinality="unspecified" mandatory="no" targetName="internalIf"/>
</vnspFolderInst>

<vnspFolderInst name="ExtConfig" scopedBy="epg" nodeNameOrLbl="N1" locked="no"
key="ExIntfConfigRelFolder"
graphNameOrLbl="FW-Graph" devCtxLbl="" ctrctNameOrLbl="Client-to-Web"
cardinality="unspecified">
<vnspCfgrRelInst name="ExtConfigrel" locked="no" key="ExIntfConfigRel" cardinality="unspecified"
mandatory="no"
targetName="externalIf"/>
</vnspFolderInst>

<vnspFolderInst name="access-list-inbound" scopedBy="epg" nodeNameOrLbl="N1" locked="no"
key="AccessList"
graphNameOrLbl="FW-Graph" devCtxLbl="" ctrctNameOrLbl="Client-to-Web"
cardinality="unspecified">
<vnspFolderInst name="permit-https" scopedBy="epg" nodeNameOrLbl="N1" locked="no"
key="AccessControlEntry"
graphNameOrLbl="FW-Graph" devCtxLbl="" ctrctNameOrLbl="Client-to-Web"
cardinality="unspecified">
<vnspParamInst name="action-permit" locked="no" key="action" cardinality="unspecified"
value="permit"
validation="" mandatory="no"/>
<vnspParamInst name="order1" locked="no" key="order" cardinality="unspecified" value="10"
validation="" mandatory="no"/>

<vnspFolderInst name="dest-service" scopedBy="epg" nodeNameOrLbl="N1" locked="no"
key="destination_service"
graphNameOrLbl="FW-Graph" devCtxLbl="" ctrctNameOrLbl="Client-to-Web"
cardinality="unspecified">
<vnspParamInst name="op" locked="no" key="operator" cardinality="unspecified" value="eq"
validation="" mandatory="no"/>
<vnspParamInst name="port" locked="no" key="low_port" cardinality="unspecified" value="https"
validation="" mandatory="no"/>
</vnspFolderInst>

```

```

<vnsFolderInst name="dest-address" scopedBy="epg" nodeNameOrLbl="N1" locked="no"
key="destination_address"
graphNameOrLbl="FW-Graph" devCtxLbl="" ctrctNameOrLbl="Client-to-Web"
cardinality="unspecified">
<vnsParamInst name="any" locked="no" key="any" cardinality="unspecified" value="any"
validation="" mandatory="no"/>
</vnsFolderInst>

<vnsFolderInst name="src-address" scopedBy="epg" nodeNameOrLbl="N1" locked="no"
key="source_address"
graphNameOrLbl="FW-Graph" devCtxLbl="" ctrctNameOrLbl="Client-to-Web"
cardinality="unspecified">
<vnsParamInst name="any" locked="no" key="any" cardinality="unspecified" value="any"
validation="" mandatory="no"/>
</vnsFolderInst>

<vnsFolderInst name="tcp" scopedBy="epg" nodeNameOrLbl="N1" locked="no" key="protocol"
graphNameOrLbl="FW-Graph" devCtxLbl="" ctrctNameOrLbl="Client-to-Web"
cardinality="unspecified">
<vnsParamInst name="tcp" locked="no" key="name_number" cardinality="unspecified" value="tcp"
validation="" mandatory="no"/>
</vnsFolderInst>
</vnsFolderInst>

<vnsFolderInst name="permit-http" scopedBy="epg" nodeNameOrLbl="N1" locked="no"
key="AccessControlEntry"
graphNameOrLbl="FW-Graph" devCtxLbl="" ctrctNameOrLbl="Client-to-Web"
cardinality="unspecified">
<vnsParamInst name="action-permit" locked="no" key="action" cardinality="unspecified"
value="permit"
validation="" mandatory="no"/>
</vnsParamInst>

<vnsParamInst name="order1" locked="no" key="order" cardinality="unspecified" value="10"
validation="" mandatory="no"/>

<vnsFolderInst name="dest-service" scopedBy="epg" nodeNameOrLbl="N1" locked="no"
key="destination_service"
graphNameOrLbl="FW-Graph" devCtxLbl="" ctrctNameOrLbl="Client-to-Web"
cardinality="unspecified">
<vnsParamInst name="op" locked="no" key="operator" cardinality="unspecified" value="eq"
validation="" mandatory="no"/>
<vnsParamInst name="port" locked="no" key="low_port" cardinality="unspecified" value="http"
validation="" mandatory="no"/>
</vnsFolderInst>

<vnsFolderInst name="dest-address" scopedBy="epg" nodeNameOrLbl="N1" locked="no"
key="destination_address"
graphNameOrLbl="FW-Graph" devCtxLbl="" ctrctNameOrLbl="Client-to-Web"
cardinality="unspecified">
<vnsParamInst name="any" locked="no" key="any" cardinality="unspecified" value="any"
validation="" mandatory="no"/>
</vnsFolderInst>

<vnsFolderInst name="src-address" scopedBy="epg" nodeNameOrLbl="N1" locked="no"
key="source_address"
graphNameOrLbl="FW-Graph" devCtxLbl="" ctrctNameOrLbl="Client-to-Web"
cardinality="unspecified">
<vnsParamInst name="any" locked="no" key="any" cardinality="unspecified" value="any"
validation="" mandatory="no"/>
</vnsFolderInst>

<vnsFolderInst name="tcp" scopedBy="epg" nodeNameOrLbl="N1" locked="no" key="protocol"
graphNameOrLbl="FW-Graph" devCtxLbl="" ctrctNameOrLbl="Client-to-Web"
cardinality="unspecified">
<vnsParamInst name="tcp" locked="no" key="name_number" cardinality="unspecified" value="tcp"
validation="" mandatory="no"/>
</vnsParamInst>
</vnsFolderInst>
</vnsFolderInst>

```

```
<fvRsProv prio="unspecified" matchT="AtleastOne" tnVzBrCPName="Client-to-Web"/>
</fvAEPg>
```

To configure the entire Layer 4 to Layer 7 ASA V firewall services for a tenant, use XML such as the following example;

```
<fvTenant ownerTag="" ownerKey="" name="Tenant-test" dn="uni/tn-Tenant-test" descr="">

<vnsLDevCtx name="" descr="" nodeNameOrLbl="N1" graphNameOrLbl="FW-Graph"
ctrctNameOrLbl="Client-to-Web"><vnsRsLDevCtxToLDev tDn="uni/tn-Tenant-test/lDevVip-ASA V"/>

<vnsLIfCtx name="" descr="" connNameOrLbl="provider">
<vnsRsLIfCtxToBD tDn="uni/tn-Tenant-test/BD-BD2"/>
<vnsRsLIfCtxToLIf tDn="uni/tn-Tenant-test/lDevVip-ASA V/lIf-provider"/>
</vnsLIfCtx>

<vnsLIfCtx name="" descr="" connNameOrLbl="consumer">
<vnsRsLIfCtxToBD tDn="uni/tn-Tenant-test/BD-BD1"/>
<vnsRsLIfCtxToLIf tDn="uni/tn-Tenant-test/lDevVip-ASA V/lIf-consumer"/>
</vnsLIfCtx>
</vnsLDevCtx>

<vzBrCP ownerTag="" ownerKey="" name="Client-to-Web" descr="" targetDscp="unspecified"
scope="tenant" prio="unspecified">

<vzSubj name="Subject" descr="" targetDscp="unspecified" prio="unspecified" revFltPorts="yes"
provMatchT="AtleastOne" consMatchT="AtleastOne">
<vzRsSubjFiltAtt tnVzFilterName="default" directives="" />
<vzRsSubjGraphAtt directives="" tnVnsAbsGraphName="FW-Graph" />
</vzSubj>
</vzBrCP>

<vnsAbsGraph ownerTag="" ownerKey="" name="FW-Graph" descr="" uiTemplateType="UNSPECIFIED">
<vnsAbsTermNodeCon ownerTag="" ownerKey="" name="T1" descr="" />
<vnsAbsTermConn ownerTag="" ownerKey="" name="1" descr="" attNotify="no" />
<vnsInTerm name="" descr="" />
<vnsOutTerm name="" descr="" />
</vnsAbsTermNodeCon>

<vnsAbsTermNodeProv ownerTag="" ownerKey="" name="T2" descr="" />
<vnsAbsTermConn ownerTag="" ownerKey="" name="1" descr="" attNotify="no" />
<vnsInTerm name="" descr="" />
<vnsOutTerm name="" descr="" />
</vnsAbsTermNodeProv>

<vnsAbsConnection ownerTag="" ownerKey="" name="C1" descr="" unicastRoute="yes"
directConnect="no" connType="external" connDir="provider" adjType="L2">
<vnsRsAbsConnectionConns
tDn="uni/tn-Tenant-test/AbsGraph-FW-Graph/AbsNode-N1/AbsFCConn-consumer"/>
<vnsRsAbsConnectionConns
tDn="uni/tn-Tenant-test/AbsGraph-FW-Graph/AbsTermNodeCon-T1/AbsTConn"/>
</vnsAbsConnection>

<vnsAbsConnection ownerTag="" ownerKey="" name="C2" descr="" unicastRoute="yes"
directConnect="no" connType="external" connDir="provider" adjType="L2">
<vnsRsAbsConnectionConns
tDn="uni/tn-Tenant-test/AbsGraph-FW-Graph/AbsNode-N1/AbsFCConn-provider"/>
<vnsRsAbsConnectionConns
tDn="uni/tn-Tenant-test/AbsGraph-FW-Graph/AbsTermNodeProv-T2/AbsTConn"/>
</vnsAbsConnection>

<vnsAbsNode ownerTag="" ownerKey="" name="N1" descr="" shareEncap="no" sequenceNumber="0"
routingMode="unspecified" managed="yes" isCopy="no" funcType="GoTo"
funcTemplateType="FW_ROUTE D">

<vnsAbsFuncConn ownerTag="" ownerKey="" name="consumer" descr="" attNotify="no">
<vnsRsMConnAtt tDn="uni/infra/mDev-CISCO-ASA-1.2/mFunc-Firewall/mConn-external" />
</vnsAbsFuncConn>

<vnsAbsFuncConn ownerTag="" ownerKey="" name="provider" descr="" attNotify="no">
```

```

<vnsRsMConnAtt tDn="uni/infra/mDev-CISCO-ASA-1.2/mFunc-Firewall/mConn-internal"/>
</vnsAbsFuncConn>
<vnsRsNodeToAbsFuncProf
tDn="uni/infra/mDev-CISCO-ASA-1.2/absFuncProfContr/absFuncProfGrp-WebServiceProfileGroup/absFuncProf-WebPolicyForRoutedMode"/>
<vnsRsNodeToLDev tDn="uni/tn-Tenant-test/lDevVip-ASAv"/>
<vnsRsNodeToMFunc tDn="uni/infra/mDev-CISCO-ASA-1.2/mFunc-Firewall"/>
</vnsAbsNode>
</vnsAbsGraph>

<fvBD ownerTag="" ownerKey="" name="BD1" descr="" unicastRoute="yes" vmac="not-applicable"
unkMcastAct="flood" unkMacUcastAct="proxy" type="regular" multiDstPktAct="bd-flood"
mcastAllow="no"
mac="00:22:BD:F8:19:FF" llAddr="::" limitIpLearnToSubnets="no" ipLearning="yes"
epMoveDetectMode="" arpFlood="no">
<fvRsBDToNdP tnNdIfPolName="" />
<fvRsCtx tnFvCtxName="VRF1" />
<fvRsIgmpsn tnIgmpSnoopPolName="" />
<fvRsBdToEpRet tnFvEpRetPolName="" resolveAct="resolve"/>
</fvBD>

<fvBD ownerTag="" ownerKey="" name="BD2" descr="" unicastRoute="yes" vmac="not-applicable"
unkMcastAct="flood" unkMacUcastAct="proxy" type="regular" multiDstPktAct="bd-flood"
mcastAllow="no"
mac="00:22:BD:F8:19:FF" llAddr="::" limitIpLearnToSubnets="no" ipLearning="yes"
epMoveDetectMode="" arpFlood="no">
<fvRsBDToNdP tnNdIfPolName="" />
<fvRsCtx tnFvCtxName="VRF1" />
<fvRsIgmpsn tnIgmpSnoopPolName="" />
<fvRsBdToEpRet tnFvEpRetPolName="" resolveAct="resolve"/>
</fvBD>

<fvCtx ownerTag="" ownerKey="" name="VRF1" descr="" pcEnfPref="enforced" pcEnfDir="ingress"
knwMcastAct="permit">
<fvRsBgpCtxPol tnBgpCtxPolName="" />
<fvRsCtxToExtRouteTagPol tnL3extRouteTagPolName="" />
<fvRsOspfCtxPol tnOspfCtxPolName="" />
<fvRsAny name="" descr="" matchT="AtleastOne" />
<fvRsCtxToEpRet tnFvEpRetPolName="" />
</fvCtx>
</vnsSvcCont/>

<fvAp ownerTag="" ownerKey="" name="ANP" descr="" prio="unspecified">
<fvAEPg name="Web" descr="" prio="unspecified" pcEnfPref="unenforced"
matchT="AtleastOne" isAttrBasedEPg="no" fwdCtrl="" />
<fvRsDomAtt tDn="uni/vmmp-VMware/dom-ACI_vDS" resImedcy="lazy" primaryEncap="unknown"
instrImedcy="lazy" encap="unknown" delimiter="" classPref="encap" />
<fvRsBd tnFvBDName="BD2" />
<fvRsCustQosPol tnQosCustomPolName="" />

<vnsFolderInst name="internalIf" nodeNameOrLbl="N1" graphNameOrLbl="FW-Graph"
ctrctNameOrLbl="Client-to-Web" scopedBy="epg" locked="no" key="Interface"
devCtxLbl="" cardinality="unspecified">
<vnsFolderInst name="internalIfCfg" nodeNameOrLbl="N1" graphNameOrLbl="FW-Graph"
ctrctNameOrLbl="Client-to-Web" scopedBy="epg" locked="no" key="InterfaceConfig"
devCtxLbl="" cardinality="unspecified">
<vnsParamInst name="internal_security_level" locked="no" key="security_level"
cardinality="unspecified"
value="100" validation="" mandatory="no" />
</vnsFolderInst>
</vnsFolderInst>

<vnsFolderInst name="externalIf" nodeNameOrLbl="N1" graphNameOrLbl="FW-Graph"
ctrctNameOrLbl="Client-to-Web" scopedBy="epg" locked="no" key="Interface"
devCtxLbl="" cardinality="unspecified">
<vnsFolderInst name="ExtAccessGroup" nodeNameOrLbl="N1" graphNameOrLbl="FW-Graph"
ctrctNameOrLbl="Client-to-Web" scopedBy="epg" locked="no" key="AccessGroup" devCtxLbl=""
cardinality="unspecified">

```

```

<vnsCfgRelInst name="name" locked="no" key="inbound_access_list_name"
cardinality="unspecified"
mandatory="no" targetName="access-list-inbound"/>
</vnsFolderInst>

<vnsFolderInst name="externalIfCfg" nodeNameOrLbl="N1" graphNameOrLbl="FW-Graph"
ctrctNameOrLbl="Client-to-Web" scopedBy="epg" locked="no" key="InterfaceConfig"
devCtxLbl="" cardinality="unspecified">
<vnsParamInst name="external_security_level" locked="no" key="security_level"
cardinality="unspecified" value="50" validation="" mandatory="no"/>
</vnsFolderInst>
</vnsFolderInst>

<vnsFolderInst name="IntConfig" nodeNameOrLbl="N1" graphNameOrLbl="FW-Graph"
ctrctNameOrLbl="Client-to-Web" scopedBy="epg" locked="no" key="InIntfConfigRelFolder"
devCtxLbl="" cardinality="unspecified">
<vnsCfgRelInst name="InConfigrel" locked="no" key="InIntfConfigRel" cardinality="unspecified"
mandatory="no" targetName="internalIf"/>
</vnsFolderInst>

<vnsFolderInst name="ExtConfig" nodeNameOrLbl="N1" graphNameOrLbl="FW-Graph"
ctrctNameOrLbl="Client-to-Web" scopedBy="epg" locked="no" key="ExIntfConfigRelFolder"
devCtxLbl="" cardinality="unspecified">
<vnsCfgRelInst name="ExtConfigrel" locked="no" key="ExIntfConfigRel" cardinality="unspecified"
mandatory="no" targetName="externalIf"/>
</vnsFolderInst>

<vnsFolderInst name="access-list-inbound" nodeNameOrLbl="N1" graphNameOrLbl="FW-Graph"
ctrctNameOrLbl="Client-to-Web" scopedBy="epg" locked="no" key="AccessList" devCtxLbl=""
cardinality="unspecified">

<vnsFolderInst name="permit-https" nodeNameOrLbl="N1" graphNameOrLbl="FW-Graph"
ctrctNameOrLbl="Client-to-Web" scopedBy="epg" locked="no" key="AccessControlEntry"
devCtxLbl="" cardinality="unspecified">
<vnsParamInst name="action-permit" locked="no" key="action" cardinality="unspecified"
value="permit" validation="" mandatory="no"/>
<vnsParamInst name="order1" locked="no" key="order" cardinality="unspecified" value="10"
validation="" mandatory="no"/>

<vnsFolderInst name="dest-service" nodeNameOrLbl="N1" graphNameOrLbl="FW-Graph"
ctrctNameOrLbl="Client-to-Web" scopedBy="epg" locked="no" key="destination_service"
devCtxLbl="" cardinality="unspecified">
<vnsParamInst name="op" locked="no" key="operator" cardinality="unspecified"
value="eq" validation="" mandatory="no"/>
<vnsParamInst name="port" locked="no" key="low_port" cardinality="unspecified"
value="https" validation="" mandatory="no"/>
</vnsFolderInst>
<vnsFolderInst name="dest-address" nodeNameOrLbl="N1" graphNameOrLbl="FW-Graph"
ctrctNameOrLbl="Client-to-Web" scopedBy="epg" locked="no" key="destination_address"
devCtxLbl="" cardinality="unspecified">
<vnsParamInst name="any" locked="no" key="any" cardinality="unspecified" value="any"
validation="" mandatory="no"/>
</vnsFolderInst>

<vnsFolderInst name="src-address" nodeNameOrLbl="N1" graphNameOrLbl="FW-Graph"
ctrctNameOrLbl="Client-to-Web" scopedBy="epg" locked="no" key="source_address"
devCtxLbl="" cardinality="unspecified">
<vnsParamInst name="any" locked="no" key="any" cardinality="unspecified" value="any"
validation="" mandatory="no"/>
</vnsFolderInst>

<vnsFolderInst name="tcp" nodeNameOrLbl="N1" graphNameOrLbl="FW-Graph"
ctrctNameOrLbl="Client-to-Web" scopedBy="epg" locked="no" key="protocol" devCtxLbl=""
cardinality="unspecified">
<vnsParamInst name="tcp" locked="no" key="name_number" cardinality="unspecified"
value="tcp" validation="" mandatory="no"/>
</vnsFolderInst>
</vnsFolderInst>

<vnsFolderInst name="permit-http" nodeNameOrLbl="N1" graphNameOrLbl="FW-Graph"
ctrctNameOrLbl="Client-to-Web" scopedBy="epg" locked="no" key="AccessControlEntry"

```

```

    devCtxLbl="" cardinality="unspecified">
<vnspParamInst name="action-permit" locked="no" key="action" cardinality="unspecified"
value="permit" validation="" mandatory="no"/>
<vnspParamInst name="order1" locked="no" key="order" cardinality="unspecified" value="10"
validation="" mandatory="no"/>

<vnspFolderInst name="dest-service" nodeNameOrLbl="N1" graphNameOrLbl="FW-Graph"
ctrctNameOrLbl="Client-to-Web" scopedBy="epg" locked="no" key="destination_service"
devCtxLbl="" cardinality="unspecified">
<vnspParamInst name="op" locked="no" key="operator" cardinality="unspecified" value="eq"
validation="" mandatory="no"/>
<vnspParamInst name="port" locked="no" key="low_port" cardinality="unspecified" value="http"
validation="" mandatory="no"/>
</vnspFolderInst>

<vnspFolderInst name="dest-address" nodeNameOrLbl="N1" graphNameOrLbl="FW-Graph"
ctrctNameOrLbl="Client-to-Web" scopedBy="epg" locked="no" key="destination_address"
devCtxLbl=""
cardinality="unspecified">
<vnspParamInst name="any" locked="no" key="any" cardinality="unspecified" value="any"
validation="" mandatory="no"/>
</vnspFolderInst>

<vnspFolderInst name="src-address" nodeNameOrLbl="N1" graphNameOrLbl="FW-Graph"
ctrctNameOrLbl="Client-to-Web" scopedBy="epg" locked="no" key="source_address" devCtxLbl=""
cardinality="unspecified">
<vnspParamInst name="any" locked="no" key="any" cardinality="unspecified" value="any"
validation="" mandatory="no"/>
</vnspFolderInst>

<vnspFolderInst name="tcp" nodeNameOrLbl="N1" graphNameOrLbl="FW-Graph"
ctrctNameOrLbl="Client-to-Web" scopedBy="epg" locked="no" key="protocol" devCtxLbl=""
cardinality="unspecified">
<vnspParamInst name="tcp" locked="no" key="name_number" cardinality="unspecified" value="tcp"
validation="" mandatory="no"/>
</vnspFolderInst>
</vnspFolderInst>
</vnspFolderInst>
<fvRsProv prio="unspecified" matchT="AtleastOne" tnVzBrCPName="Client-to-Web"/>
</fvAEPg>

<fvAEPg name="Client" descr="" prio="unspecified" pcEnfPref="unenforced" matchT="AtleastOne"
isAttrBasedEPg="no" fwdCtrl="">
<fvRsCons prio="unspecified" tnVzBrCPName="Client-to-Web"/>
<fvRsDomAtt tDn="uni/vmmp-VMware/dom-ACI_vDS" resImedcy="lazy" primaryEncap="unknown"
instrImedcy="lazy" encap="unknown" delimiter="" classPref="encap"/>
<fvRsBd tnFvBDName="BD1"/>
<fvRsCustQosPol tnQosCustomPolName="" />
</fvAEPg>
</fvAp>
<fvRsTenantMonPol tnMonEPGPolName="" />

<vnslDevVip name="ASAv" managed="yes" isCopy="no" funcType="GoTo" trunking="no"
svcType="FW" packageModel="ASAv" mode="legacy-Mode" devtype="VIRTUAL"
contextAware="single-Context">
<vnscCred name="username" value="admin"/>
<vnscRsMDevAtt tDn="uni/infra/mDev-CISCO-ASA-1.2"/>
<vnscCredSecret name="password"/>
<vnscCMgmt name="" port="443" host="172.31.184.249"/>
<vnscRsALDevToDomP tDn="uni/vmmp-VMware/dom-ACI_vDS"/>

<vnscDev name="Device1" devCtxLbl="" vmName="ASAv-L3" vcenterName="vcenter">
<vnscCred name="username" value="admin"/>
<vnscCredSecret name="password"/>
<vnscCMgmt name="" port="443" host="172.31.184.249"/>
<vnscCIf name="GigabitEthernet0/1" vnicName="Network adapter 3"/>
<vnscCIf name="GigabitEthernet0/0" vnicName="Network adapter 2"/>
<vnscRsCDevToCtrlrP tDn="uni/vmmp-VMware/dom-ACI_vDS/ctrlr-vcenter"/>

```

```

</vnsCDev>

<vnsLIf name="provider" encaps="unknown">
<vnsRsMetaIf tDn="uni/infra/mDev-CISCO-ASA-1.2/mIfLbl-internal" isConAndProv="no"/>
<vnsRsCIfAttN tDn="uni/tn-Tenant-test/lDevVip-ASAv/cDev-Device1/cIf-[GigabitEthernet0/1]"/>
</vnsLIf>

<vnsLIf name="consumer" encaps="unknown">
<vnsRsMetaIf tDn="uni/infra/mDev-CISCO-ASA-1.2/mIfLbl-external" isConAndProv="no"/>
<vnsRsCIfAttN tDn="uni/tn-Tenant-test/lDevVip-ASAv/cDev-Device1/cIf-[GigabitEthernet0/0]"/>
</vnsLIf>
</vnsLDevVip>
</fvTenant>

```

Example: Configuring Layer 4 to Layer 7 Route Peering

Configuring Layer 4 to Layer 7 Route Peering With the REST API

These `l3extOut` policies specify the OSPF configurations needed to enable OSPF on the fabric leaf and are very similar to the `l3extOut` policies used for external communication.

The `l3extOut` policies also specify the prefix-based EPGs that control which routes are distributed in/out of the fabric. The `scope=import` attribute controls two things: which endpoint prefixes are learned; and directs the external L4-L7 device to advertise this route. The `scope=export` attribute specifies that the fabric has to advertise this route to the L4-L7 device.

Two sample `l3extOut` policies are shown below: `OspfInternal` deployed on `eth1/23`, and `OspfExternal` deployed on `eth1/25`.

Before You Begin

Create one or more `l3extOut` external network connections and deploy them on the fabric leaf nodes where the service device is connected.

Procedure

-
- Step 1** To configure `OspfInternal` on `eth1/23`, send a post with XML similar to the following example:

Example:

```

<?xml version="1.0" encoding="UTF-8?>
<!-- /api/policymgr/mo.xml -->
<polUni>
    <fvTenant name="coke{{tenantId}}">
        {%
            if status is not defined %
                {%
                    set status = "created,modified" %
                }
            endif %
        }

        <l3extOut name="OspfInternal" status="{{status}}">

            <l3extLNodeP name="bLeaf-101">
                <l3extRsNodeL3OutAtt tDn="topology/pod-1/node-101" rtrId="180.0.0.11"/>
                <l3extLifP name='portIf'>
                    <l3extRsPathL3OutAtt tDn="topology/pod-1/paths-101/pathEp-[eth1/23]" ifInstT='ext-svi' encaps='vlan-3844' addr="30.30.30.100/28" mtu='1500' />

                    <!-- <ospfIfP authKey="tecom" authType="md5" authKeyId='1'> -->
                    <ospfIfP>
                        <ospfRsIfPol tnOspfIfPolName='ospfIfPol' />
                    </ospfIfP>
                </l3extLifP>
            </l3extLNodeP>
        </l3extOut>
    </fvTenant>
</polUni>

```

```

</l3extLNodeP>

<ospfExtP areaId='111' areaType='nssa' areaCtrl='redistribute'>

    <l3extInstP name="OspfInternalInstP">
        <l3extSubnet ip="30.30.30.100/28" scope="import"/>
        <l3extSubnet ip="20.20.20.0/24" scope="import"/>
        <l3extSubnet ip="10.10.10.0/24" scope="export"/>
    </l3extInstP>

    <l3extRsEctx tnFvCtxName="cokectx1"/>

</l3extOut>

<ospfIfPol name="ospfIfPol" nwT='bcast' xmitDelay='1'
            helloIntvl='10' deadIntvl='40' status="created,modified"/>
</fvTenant>
</polUni>

```

Step 2 To configure OspfExternal on eth1/25, send a post with XML similar to the following example:

Example:

```

<?xml version="1.0" encoding="UTF-8?>
<!-- /api/policymgr/mo.xml -->
<polUni>
    <fvTenant name="common">
        <fvCtx name="commonctx"/>

        {%
            if status is not defined %
                {%
                    set status="created,modified" %
                }
            {%
                endif %
            }

        <l3extOut name=OspfExternal" status="{{status}}">
            <l3extLNodeP name="bLeaf-101">
                <l3extRsNodeL3OutAtt tDn="topology/pod-1/node-101" rtrId="180.0.0.8/28"/>
                <l3extLIfP name='portIf'>
                    {%
                        if intfType is not defined %
                            {%
                                set intfType="ext-svi" %
                            }
                        {%
                            endif %
                        }

                    <l3extRsPathL3OutAtt tDn="topology/pod-1/pathes-101/pathep-[eth1/25]"
                        ifInstT='ext-svi' encap='vlan-3843' addr="40.40.40.100/28" mtu='1500' />
                    <!-- ospfIfP authKey="tecom" authType="md5" authKeyId='1' -->
                    <ospfIfP>
                        <ospfRsIfPol tnOspfIfPolName='ospfIfPol' />
                    </ospfIfP>
                </l3extLIfP>
            </l3extLNodeP>
        </l3extOut>

        <ospfExtP areaId='111' areaType='nssa' areaCtrl='redistribute'>

            <l3extInstP name="OspfExternalInstP">
                <l3extSubnet ip="40.40.40.100/28" scope="import"/>
                <l3extSubnet ip="10.10.10.0/24" scope="import"/>
                <l3extSubnet ip="20.20.20.0/24" scope="export"/>
            </l3extInstP>

            <l3extRsEctx tnFvCtxName="commonctx"/>

        </l3extOut>

        <ospfIfPol name="ospfIfPol" nwT='bcast' xmitDelay='1' helloIntvl='10' deadIntvl='40'
                    status="created,modified"/>
    </fvTenant>
</polUni>

```

The `l3extInstP` object specifies that prefixes 40.40.40.100/28 and 10.10.10.0/24 are to be used for prefix based endpoint association and indicate that the L4-L7 device should advertise these routes.

The `l3extRsPathL3OutAtt` object specifies where each `L3extOut` is deployed.

Note For route peering to work, the `l3extRsPathL3OutAtt` must match the `RsCIfPathAtt` where the L4-L7 logical device cluster is connected.

Specifying an `L3extOut` Policy for Layer 4 to L7 Route Peering

A specific `l3extOut` policy can be used for a logical device cluster using its selection policy `vnsLIfCtx`. The `vnsRsLIfCtxToInstP` points the `LIfCtx` to the appropriate `OspfInternal` and `OspfExternal` `l3extInstP` EPGs. To configure an `L3extOut` policy used for Layer 4 to Layer 7 Route Peering, send a post with XML such as the following example:

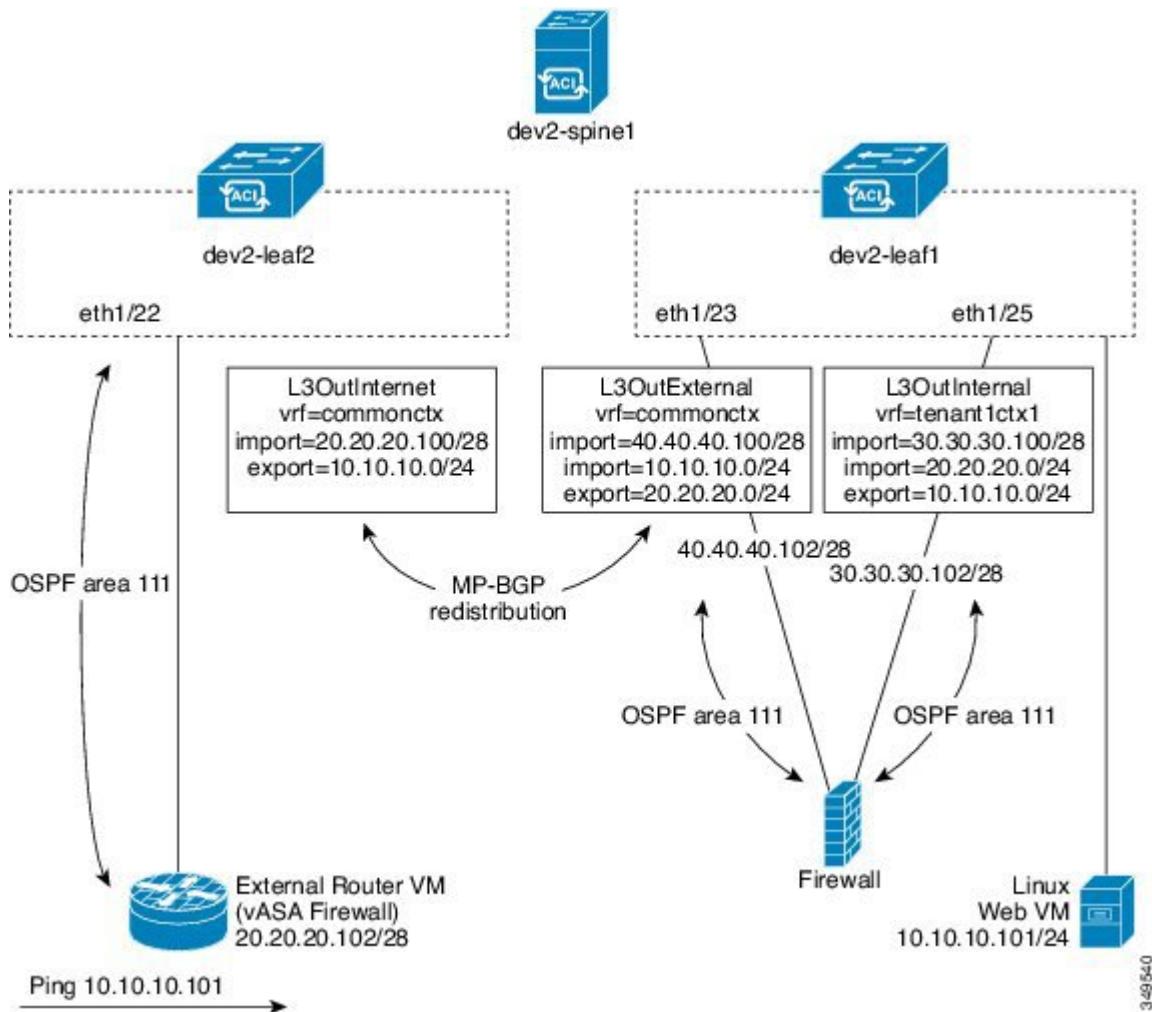
```
<vnsLDevCtx crctNameOrLbl="webCtrct{{graphId}}" graphNameOrLbl="WebGraph" nodeNameOrLbl="FW">
    <vnsRsLDevCtxToLDev tDn="uni/tn-solar{{tenantId}}/lDevVip-Firewall"/>
    <vnsLIfCtx connNameOrLbl="internal">
        {%
            if L3ExtOutInternal is not defined %
        <fvSubnet ip="10.10.10.10/24"/>
        {%
            endif %
        <vnsRsLIfCtxToBD tDn="uni/tn-solar{{tenantId}}/BD-solarBD1"/>
        <vnsRsLIfCtxToLIf tDn="uni/tn-solar{{tenantId}}/lDevVip-Firewall/lif-internal"/>
        {%
            if L3ExtOutInternal is defined %
        <vnsRsLIfCtxToInstP
            tDn="uni/tn-solar{{tenantId}}/out-OspfInternal/instP-OspfInternalInstP"
            status="{{L3ExtOutInternal}}"/>
            {%
                endif %
            </vnsLIfCtx>
            <vnsLIfCtx connNameOrLbl="external">
                {%
                    if L3ExtOutExternal is not defined %
                <fvSubnet ip="40.40.40.40/24"/>
                {%
                    endif %
                <vnsRsLIfCtxToBD tDn="uni/tn-solar{{tenantId}}/BD-solarBD4"/>
                <vnsRsLIfCtxToLIf tDn="uni/tn-solar{{tenantId}}/lDevVip-Firewall/lif-external"/>
                {%
                    if L3ExtOutExternal is defined %
                <vnsRsLIfCtxToInstP
                    tDn="uni/tn-solar{{tenantId}}/out-OspfExternal/instP-OspfExternalInstP"
                    status="{{L3ExtOutExternal}}"/>
                    {%
                        endif %
                    </vnsLIfCtx>
                </vnsLDevCtx>
            </vnsLDevCtx>
```

The associated concrete device needs to have a `vnsRsCIfPathAtt` that deploys it to the same fabric leaf as shown in the following example:

```
<vnsCDev name="ASA">
    <vnsRsLDevCtxToLDev tDn="uni/tn-solar{{tenantId}}/lDevVip-Firewall"/>
    <vnsCIf name="Gig0/0">
        <vnsRsCIfPathAtt tDn="topology/pod-1/paths-101/pathep-[eht1/23]"/>
    </vnsCIf>
    <vnsCIf name="Gig0/1">
        <vnsRsCIfPathAtt tDn="topology/pod-1/paths-101/pathep-[eht1/25]"/>
    </vnsCIf>
    <vnsCMgmt name="devMgmt" host="{{asaIp}}" port="443" />
    <vnsCCred name="username" value="admin" />
    <vnsCCredSecret name="password" value="insieme" />
</vnsCDev>
```

The following figure shows how route peering works end-to-end.

Figure 36: Sample Deployment



In this 2-leaf, 1-spine topology, the linux web server is at IP 10.10.10.101/24 and is hosted on an ESX server connected to dev2-leaf1. A service graph is deployed consisting of a two-arm firewall that is also connected to dev2-leaf1. The service graph is associated with a contract that binds an external `l3extOut` L3OutInternet with the provider EPG (Web VM). Two internal `l3extOut` policies, an L3OutExternal, and an L3OutInternal are also deployed to the leaf ports where the service device is connected.



CHAPTER 15

Configuring QoS

- [CoS Preservation, page 313](#)
- [Multipod QoS, page 315](#)
- [Translating QoS Ingress Markings to Egress Markings, page 316](#)

CoS Preservation

Preserving 802.1P Class of Service Settings

APIC enables preserving 802.1P class of service (CoS) settings within the fabric. Enable the fabric global QoS policy `dot1p-preserve` option to guarantee that the CoS value in packets which enter and transit the ACI fabric is preserved.

802.1P CoS preservation is supported in single pod and multipod topologies.

In multipod topologies, CoS Preservation can be used where you want to preserve the QoS priority settings of 802.1P traffic entering POD 1 and egressing out of POD 2, but you are not concerned with preserving the CoS/DSCP settings in interpod network (IPN) traffic between the pods. To preserve CoS/DSCP settings when multipod traffic is transiting an IPN, use a DSCP policy. For more information, see [Preserving QoS Priority Settings in a Multipod Fabric, on page 315](#).

Observe the following 801.1P CoS preservation guidelines and limitations:

- The current release can only preserve the 802.1P value within a VLAN header. The DEI bit is not preserved.
- For VXLAN encapsulated packets, the current release will not preserve the 802.1P CoS value contained in the outer header.
- 802.1P is not preserved when the following configuration options are enabled:
 - Multipod QoS (using a DSCP policy) is enabled.
 - Contracts are configured that include QoS.
 - Dynamic packet prioritization is enabled.
 - The outgoing interface is on a FEX.

- Preserving QoS CoS priority settings is not supported when traffic is flowing from an EPG with isolation enforced to an EPG without isolation enforced.
- A DSCP QoS policy is configured on a VLAN EPG and the packet has an IP header. DSCP marking can be set at the filter level on the following with the precedence order from the innermost to the outermost:
 - Contract
 - Subject
 - In Term
 - Out Term

**Note**

When specifying vzAny for a contract, external EPG DSCP values are not honored because vzAny is a collection of all EPGs in a VRF, and EPG specific configuration cannot be applied. If EPG specific target DSCP values are required, then the external EPG should not use vzAny.

Preserving QoS CoS Settings Using the REST API

Procedure

Step 1 Enable CoS preservation, using a REST API POST statement, similar to the following:

Example:

```
post https://192.0.20.123/api/node/mo/uni/infra/qosinst-default.xml
<imdata totalCount="1">
<qosInstPol ownerTag="" ownerKey="" name="default" dn="uni/infra/qosinst-default" descr=""
ctrl="dot1p-preserve"/>
</imdata>
```

Step 2 Disable CoS preservation, using a POST statement, such as the following example, which leaves the `ctrl` property empty:

Example:

```
post https://192.0.20.123/api/node/mo/uni/infra/qosinst-default.xml
<imdata totalCount="1">
<qosInstPol ownerTag="" ownerKey="" name="default" dn="uni/infra/qosinst-default" descr=""
ctrl=""/>
</imdata>
```

Multipod QoS

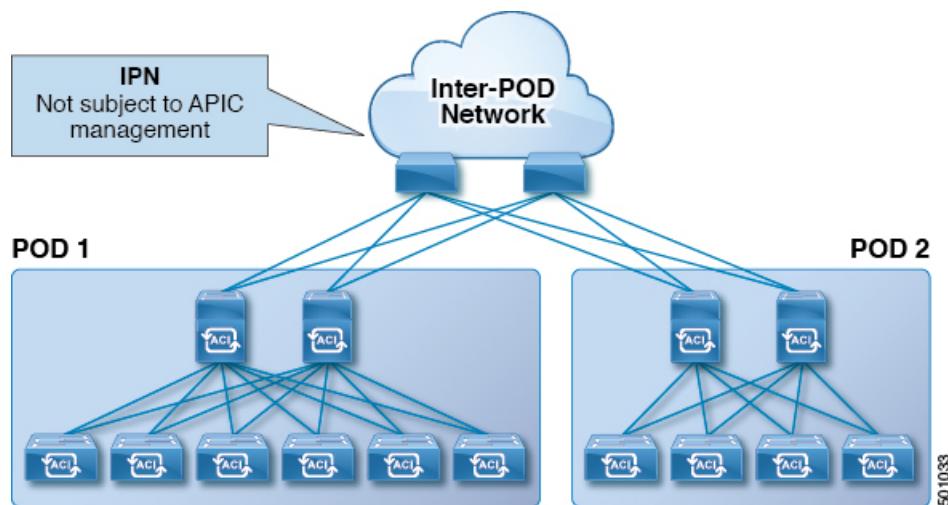
Preserving QoS Priority Settings in a Multipod Fabric

This topic describes how to guarantee QoS priority settings in a multipod topology, where devices in the interpod network are not under APIC management, and may modify 802.1P settings in traffic transiting their network.


Note

You can alternatively use CoS Preservation where you want to preserve the QoS priority settings of 802.1P traffic entering POD 1 and egressing out of POD 2, but you are not concerned with preserving the CoS/DSCP settings in interpod network (IPN) traffic between the pods. For more information, see [Preserving 802.1P Class of Service Settings, on page 313](#).

Figure 37: Multipod Topology



As illustrated in this figure, traffic between pods in a multipod topology passes through an IPN, which may not be under APIC management. When an 802.1P frame is sent from a spine or leaf switch in POD 1, the devices in the IPN may not preserve the CoS setting in 802.1P frames. In this situation, when the frame reaches a POD 2 spine or leaf switch, it has the CoS level assigned by the IPN device, instead of the level assigned at the source in POD 1. Use a DSCP policy to ensure that the QoS priority levels are preserved in this case.

Configure a DSCP policy to preserve the QoS priority settings in a multipod topology, where there is a need to do deterministic mapping from CoS to DSCP levels for different traffic types, and you want to prevent the devices in the IPN from changing the configured levels. With a DSCP policy enabled, APIC converts the CoS level to a DSCP level, according to the mapping you configure. When a frame is sent from POD 1 (with the PCP level mapped to a DSCP level), when it reaches POD 2, the mapped DSCP level is then mapped back to the original PCP CoS level.

Creating a DSCP Policy Using the REST API

Procedure

Step 1 Configure and enable a DSCP policy with a post, such as the following:

Example:

```
post https://192.0.20.123/api/node/mo/uni/tn-infra/dscptranspol-default.xml

<imdata totalCount="1">

<qosDscpTransPol traceroute="AF43" span="AF42" policy="AF22" ownerTag="" ownerKey=""
name="default"
level3="AF13" level2="AF12" level1="AF11" dn="uni/tn-infra/dscptranspol-default" descr=""
control="AF21" adminSt="enabled"/>

</imdata>
```

Step 2 Disable the DSCP policy with a post such as the following:

Example:

```
post https://192.0.20.123/api/node/mo/uni/tn-infra/dscptranspol-default.xml

<imdata totalCount="1">

<qosDscpTransPol traceroute="AF43" span="AF42" policy="AF22" ownerTag="" ownerKey=""
name="default"
level3="AF13" level2="AF12" level1="AF11" dn="uni/tn-infra/dscptranspol-default" descr=""
control="AF21" adminSt="disabled"/>

</imdata>
```

Translating QoS Ingress Markings to Egress Markings

Translating QoS Ingress Markings to Egress Markings

APIC enables translating the 802.1P CoS field (Class of Service) based on the ingress DSCP value. 802.1P CoS translation is supported only if DSCP is present in the IP packet and dot1P is present in the Ethernet frames.

This functionality enables the ACI Fabric to classify the traffic for devices that classify the traffic based only on the CoS value. It allows mapping the dot1P CoS value based on the ingress dot1P value. It is mainly applicable for Layer 2 packets, which do not have an IP header.

Observe the following 802.1P CoS translation guidelines and limitations:

- Enable the fabric global QoS policy `dot1p-preserve` option.
- 802.1P CoS translation is not supported on external L3 interfaces.
- 802.1P CoS translation is supported only if the egress frame is 802.1Q encapsulated.

802.1P CoS translation is not supported when the following configuration options are enabled:

- Contracts are configured that include QoS.
- The outgoing interface is on a FEX.
- Multipod QoS using a DSCP policy is enabled.
- Dynamic packet prioritization is enabled.
- If an EPG is configured with intra-EPG endpoint isolation enforced.
- If an EPG is configured with allow-microsegmentation enabled.

Translating QoS Ingress Markings to Egress Markings Using the REST API

Create a custom QoS policy and then associate the policy with an EPG.

Before You Begin

Create the tenant, application, and EPGs that will consume the custom QoS policy. The example creates the `vrfQos001` custom QoS policy and associates it with the `ep2` EPG, that will consume it.

Procedure

Step 1 Create a custom QoS policy by sending a post with XML such as the following example:

Example:

```
<qosCustomPol name="vrfQos001" dn="uni/tn-t001/qoscustom-vrfQos001">
  <qosDscpClass to="AF31" targetCos="6"
    target="unspecified" prio="unspecified" from="AF23"/>
  <qosDot1PClass to="1" targetCos="6" target="unspecified"
    prio="unspecified" from="0"/>
</qosCustomPol>
```

Step 2 Associate the policy with an EPG that will consume it by sending a post with XML such as the following example:

Example:

```
<fvAEPg
  prio="unspecified" prefGrMemb="exclude" pcEnfPref="unenforced"
  name="ep2" matchT="AtleastOne" isAttrBasedEPg="no" fwdCtrl=""
  dn="uni/tn-t001/ap-ap2/epg-ep2">
  <fvRsDomAtt tDn="uni/vmmp-VMware/dom-vs1" resImedcy="lazy"
    primaryEncap="unknown" netflowPref="disabled" instrImedcy="lazy" encapMode="auto"
    encap="unknown" delimiter="" classPref="encap"/>
  <fvRsCustQosPol tnQosCustomPolName="vrfQos001"/>
  <fvRsBd tnFvBDName="default"/>
</fvAEPg>
```

Troubleshooting Cisco APIC QoS Policies

The following table summarizes common troubleshooting scenarios for the Cisco APIC QoS.

Problem	Solution
Unable to update a configured QoS policy.	<p>1 Invoke the following API to ensure that <code>qospDscpRule</code> is present on the leaf.</p> <p>GET <code>https://192.0.20.123/api/node/class/qospDscpRule.xml</code></p> <p>2 Ensure that the QoS rules are accurately configured and associated to the EPG ID to which the policy is attached.</p> <p>Use the following NX-OS style CLI commands to verify the configuration.</p> <pre>leaf1# show vlan leaf1# show system internal aclqos qos policy detail apic1# show running-config tenant tenant-name policy-map type qos custom-qos-policy-name apic1# show running-config tenant tenant-name application application-name epg epg-name</pre>



CHAPTER 16

Configuring Security

- Enabling TACACS+, RADIUS, and LDAP, page 319
- Configuring FIPS, page 322
- Configuring Fabric Secure Mode, page 323
- Enabling RBAC, page 324
- Enabling Port Security, page 340
- Enabling COOP Authentication, page 342

Enabling TACACS+, RADIUS, and LDAP

Overview

This article provides step by step instructions on how to enable RADIUS, TACACS+, and LDAP users access the APIC. It assumes the reader is thoroughly familiar with the Cisco Application Centric Infrastructure Fundamentals manual, especially the User Access, Authentication, and Accounting chapter.

Guidelines

When configuring an external authentication server to access the Cisco APIC, follow these guidelines:

- Whenever a .
- While configuring .
- By definition, .
- This configuration task is applicable for .
- The user must configure .
- The autonomous system feature can only be .

Configuring APIC for TACACS+ Using the REST API

- The Cisco Application Centric Infrastructure (ACI) fabric must be installed, Application Policy Infrastructure Controllers (APICs) must be online, and the APIC cluster must be formed and healthy.
- The TACACS+ server host name or IP address, port, and key must be available.
- The APIC management endpoint group must be available.

Procedure

Step 1 Configure the TACACS+ Provider by sending a POST request with XML such as the following example:

Example:

```
<aaaTacacsPlusProvider timeout="5" retries="1" port="49" name="192.168.200.1"
monitoringUser="test" monitorServer="disabled"
dn="uni/userext/tacacsexst/tacacsplusprovider-192.168.200.1" authProtocol="pap"/>
```

Step 2 Configure the TACACS+ Provider Group by sending a POST request with XML such as the following example:

Example:

```
<aaaTacacsPlusProviderGroup name="TENANT64_TACACS_provGrp"
dn="uni/userext/tacacsexst/tacacsplusprovidergroup-TENANT64_TACACS_provGrp"/>
```

Step 3 Configure the TACACS+ Login Domain by sending a POST request with XML such as the following example:

Example:

```
<aaaLoginDomain name="TENANT64_TACACS_LoginDom"
dn="uni/userext/logindomain-TENANT64_TACACS_LoginDom"/>
```

The entire configuration can be sent in one POST request, with XML such as this example:

```
<aaaTacacsPlusProvider timeout="5" retries="1" port="49"
name="192.168.200.1" monitoringUser="test" monitorServer="disabled"
dn="uni/userext/tacacsexst/tacacsplusprovider-192.168.200.1" authProtocol="pap"/>
<aaaTacacsPlusProviderGroup name="TENANT64_TACACS_provGrp"
dn="uni/userext/tacacsexst/tacacsplusprovidergroup-TENANT64_TACACS_provGrp"/>
<aaaLoginDomain name="TENANT64_TACACS_LoginDom"
dn="uni/userext/logindomain-TENANT64_TACACS_LoginDom"/>
```

Configuring APIC for RADIUS Using the REST API

Before You Begin

- The ACI fabric must be installed, Application Policy Infrastructure Controllers (APICs) must be online, and the APIC cluster must be formed and healthy.
- The RADIUS server host name or IP address, port, authorization protocol, and key must be available.
- The APIC management endpoint group must be available.

Procedure

- Step 1** Configure the RADIUS Provider by sending a POST request with XML such as the following example:

Example:

```
<aaaRadiusProvider timeout="5" retries="1" name="TENANT64_RADIUS-host.com"
monitoringUser="test" monitorServer="disabled"
dn="uni/userext/radiusext/radiusprovider-TENANT64_RADIUS-host.com" authProtocol="pap"
authPort="1812"/>
```

- Step 2** Configure the RADIUS Provider Group by sending a POST request with XML such as the following example:

Example:

```
<aaaRadiusProviderGroup name="TENANT64_RADIUS_provGrp"
dn="uni/userext/radiusext/radiusprovidergroup-TENANT64_RADIUS_provGrp"/>
```

- Step 3** Configure the RADIUS Login Domain by sending a POST request with XML such as the following example:

Example:

```
<aaaLoginDomain name="TENANT64_RADIUSLoginDom"
dn="uni/userext/logindomain-TENANT64_RADIUSLoginDom"/>
```

The entire configuration can be sent as one POST request, with XML such as this example:

```
<aaaRadiusProvider
timeout="5" retries="1" name="TENANT64_RADIUS-host.com" monitoringUser="test"
monitorServer="disabled"
dn="uni/userext/radiusext/radiusprovider-TENANT64_RADIUS-host.com" authProtocol="pap"
authPort="1812"/>
<aaaRadiusProviderGroup
name="TENANT64_RADIUS_provGrp"
dn="uni/userext/radiusext/radiusprovidergroup-TENANT64_RADIUS_provGrp"/>
<aaaLoginDomain
name="TENANT64_RADIUSLoginDom" dn="uni/userext/logindomain-TENANT64_RADIUSLoginDom"/>
```

Configuring APIC for LDAP Using the REST API

Before You Begin

- The Cisco Application Centric Infrastructure (ACI) fabric must be installed, Application Policy Infrastructure Controllers (APICs) must be online, and the APIC cluster must be formed and healthy.
- The LDAP server host name or IP address, port, bind DN, Base DN, and password must be available.
- The APIC management endpoint group must be available.

Procedure

- Step 1** Configure the LDAP Provider by sending a POST request with XML such as the following example:

Example:

```
<aaaLdapProvider timeout="30" rootdn="" retries="1" port="389" name="TENANT64_LDAP-host.com"
monitoringUser="test" monitorServer="disabled" filter="cn=$userid" enableSSL="yes"
```

```
dn="uni/userext/ldapext/ldaprovider-TENANT64_LDAP-host.com" descr="" basedn=""
attribute="CiscoAVPair" SSLValidationLevel="strict"/>
```

Step 2 Configure the LDAP Provider Group by sending a POST request with XML such as the following example:

Example:

```
<aaaLdapProviderGroup name="TENANT64_LDAP-ProvGrp"
dn="uni/userext/ldapext/ldaprovidergroup-TENANT64_LDAP-ProvGrp"/>
```

Step 3 Configure the LDAP Login Domain by sending a POST request with XML such as the following example:

Example:

```
<aaaDomainAuth realm="ldap" providerGroup="TENANT64_LDAP-ProvGrp"
dn="uni/userext/logindomain-TENANT64_LDAPLoginDom/domainauth"/>
```

The entire configuration can be sent in one POST request, with XML such as the following example:

```
<aaaLdapProvider
timeout="30" rootdn="" retries="1" port="389" name="TENANT64_LDAP-host.com"
monitoringUser="test" monitorServer="disabled" filter="cn=$userid" enableSSL="yes"
dn="uni/userext/ldapext/ldaprovider-TENANT64_LDAP-host.com" descr="" basedn=""
attribute="CiscoAVPair" SSLValidationLevel="strict"/>
<aaaLdapProviderGroup
name="TENANT64_LDAP-ProvGrp" dn="uni/userext/ldapext/ldaprovidergroup-TENANT64_LDAP-ProvGrp"/>
<aaaDomainAuth
realm="ldap" providerGroup="TENANT64_LDAP-ProvGrp"
dn="uni/userext/logindomain-TENANT64_LDAPLoginDom/domainauth"/>
```

Configuring FIPS

About Federal Information Processing Standards (FIPS)

The Federal Information Processing Standards (FIPS) Publication 140-2, Security Requirements for Cryptographic Modules, details the U.S. government requirements for cryptographic modules. FIPS 140-2 specifies that a cryptographic module should be a set of hardware, software, firmware, or some combination that implements cryptographic functions or processes, including cryptographic algorithms and, optionally, key generation, and is contained within a defined cryptographic boundary.

FIPS specifies certain cryptographic algorithms as secure, and it also identifies which algorithms should be used if a cryptographic module is to be called FIPS compliant.

Guidelines and Limitations

Follow these guidelines and limitations:

- When FIPS is enabled, it is applied across Cisco APIC.
- When performing a Cisco APIC software downgrade, you must disable FIPS first.
- Make your passwords a minimum of eight characters in length.
- Disable Telnet. Users should log in using SSH only.
- Delete all SSH Server RSA1 keypairs.
- Disable remote authentication through RADIUS/TACACS+. Only local and LDAP users can be authenticated.

- Secure Shell (SSH) and SNMP are supported.
- Disable SNMP v1 and v2. Any existing user accounts on the switch that have been configured for SNMPv3 should be configured only with SHA for authentication and AES/3DES for privacy.
- Starting with release 2.2(x), FIPS can be configured at the switch level.

Configuring FIPS for Cisco APIC Using REST API

When FIPS is enabled, it is applied across Cisco APIC.

Procedure

Configure FIPS for all tenants.

Example:

```
https://apic1.cisco.com/api/node/mo/uni/userext.xml
<aaaFabricSec fipsMode="enable" />
```

Note You must reboot to complete the configuration. Anytime you change the mode, you must reboot to complete the configuration.

Configuring Fabric Secure Mode

Fabric Secure Mode

Fabric secure mode prevents parties with physical access to the fabric equipment from adding a switch or APIC controller to the fabric without manual authorization by an administrator. Starting with release 1.2(1x), the firmware checks that switches and controllers in the fabric have valid serial numbers associated with a valid Cisco digitally signed certificate. This validation is performed upon upgrade to this release or during an initial installation of the fabric. The default setting for this feature is permissive mode; an existing fabric continues to run as it has after an upgrade to release 1.2(1) or later. An administrator with fabric-wide access rights must enable strict mode. The following table summarizes the two modes of operation:

Permissive Mode (default)	Strict Mode
Allows an existing fabric to operate normally even though one or more switches have an invalid certificate.	Only switches with a valid Cisco serial number and SSL certificate are allowed.
Does not enforce serial number based authorization.	Enforces serial number authorization.
Allows auto-discovered controllers and switches to join the fabric without enforcing serial number authorization.	Requires an administrator to manually authorize controllers and switches to join the fabric.

Configuring Fabric Secure Mode Using the REST API

To manage Secure Fabric Mode using the REST API, perform the following steps:

Procedure

- Step 1** To enable strict mode, send a POST request with XML such as the following example:

Example:

```
POST https://apic-ip-address/api/node/mo/uni.xml?
<pkiFabricCommunicationEp mode="strict"/>
```

- Step 2** To enable permissive mode, send a POST request with XML such as the following example:

Example:

```
POST https://apic-ip-address/api/node/mo/uni.xml?
<pkiFabricCommunicationEp mode="permissive"/>
```

- Step 3** To authorize a controller, send a POST request with XML such as the following example:

Example:

```
POST https://apic-ip-address/api/mo/uni/controller.xml?
<fabricNodeIdentPol>
  <fabricCtrlrIdentP serial="TEP-1-1"/>
</fabricNodeIdentPol>
```

- Step 4** To reject a controller, send a POST request with XML such as the following example:

Example:

```
POST https://apic-ip-address/api/mo/uni/controller.xml?
<fabricNodeIdentPol>
  <fabricCtrlrIdentP serial="FCH1750V025" reject="yes"/>
</fabricNodeIdentPol>
```

Enabling RBAC

Access Rights Workflow Dependencies

The Cisco Application Centric Infrastructure (ACI) RBAC rules enable or restrict access to some or all of the fabric. For example, in order to configure a leaf switch for bare metal server access, the logged in administrator must have rights to the `infra` domain. By default, a tenant administrator does not have rights to the `infra` domain. In this case, a tenant administrator who plans to use a bare metal server connected to a leaf switch could not complete all the necessary steps to do so. The tenant administrator would have to coordinate with a fabric administrator who has rights to the `infra` domain. The fabric administrator would set up the switch configuration policies that the tenant administrator would use to deploy an application policy that uses the bare metal server attached to an ACI leaf switch.

User Access, Authorization, and Accounting

Application Policy Infrastructure Controller (APIC) policies manage the authentication, authorization, and accounting (AAA) functions of the Cisco Application Centric Infrastructure (ACI) fabric. The combination of user privileges, roles, and domains with access rights inheritance enables administrators to configure AAA functions at the managed object level in a granular fashion. These configurations can be implemented using the REST API, the CLI, or the GUI.

Multiple Tenant Support

A core Application Policy Infrastructure Controller (APIC) internal data access control system provides multitenant isolation and prevents information privacy from being compromised across tenants. Read/write restrictions prevent any tenant from seeing any other tenant's configuration, statistics, faults, or event data. Unless the administrator assigns permissions to do so, tenants are restricted from reading fabric configuration, policies, statistics, faults, or events.

User Access: Roles, Privileges, and Security Domains

The APIC provides access according to a user's role through role-based access control (RBAC). An Cisco Application Centric Infrastructure (ACI) fabric user is associated with the following:

- A set of roles
- For each role, a privilege type: no access, read-only, or read-write
- One or more security domain tags that identify the portions of the management information tree (MIT) that a user can access

The ACI fabric manages access privileges at the managed object (MO) level. A privilege is an MO that enables or restricts access to a particular function within the system. For example, fabric-equipment is a privilege bit. This bit is set by the Application Policy Infrastructure Controller (APIC) on all objects that correspond to equipment in the physical fabric.

A role is a collection of privilege bits. For example, because an "admin" role is configured with privilege bits for "fabric-equipment" and "tenant-security," the "admin" role has access to all objects that correspond to equipment of the fabric and tenant security.

A security domain is a tag associated with a certain subtree in the ACI MIT object hierarchy. For example, the default tenant "common" has a domain tag `common`. Similarly, the special domain tag `all` includes the entire MIT object tree. An administrator can assign custom domain tags to the MIT object hierarchy. For example, an administrator could assign the "solar" domain tag to the tenant named solar. Within the MIT, only certain objects can be tagged as security domains. For example, a tenant can be tagged as a security domain but objects within a tenant cannot.

Creating a user and assigning a role to that user does not enable access rights. It is necessary to also assign the user to one or more security domains. By default, the ACI fabric includes two special pre-created domains:

- `All`—allows access to the entire MIT
- `Infra`— allows access to fabric infrastructure objects/subtrees, such as fabric access policies

**Note**

For read operations to the managed objects that a user's credentials do not allow, a "DN/Class Not Found" error is returned, not "DN/Class Unauthorized to read." For write operations to a managed object that a user's credentials do not allow, an HTTP 401 Unauthorized error is returned. In the GUI, actions that a user's credentials do not allow, either they are not presented, or they are grayed out.

A set of predefined managed object classes can be associated with domains. These classes should not have overlapping containment. Examples of classes that support domain association are as follows:

- Layer 2 and Layer 3 network managed objects
- Network profiles (such as physical, Layer 2, Layer 3, management)
- QoS policies

When an object that can be associated with a domain is created, the user must assign domain(s) to the object within the limits of the user's access rights. Domain assignment can be modified at any time.

If a virtual machine management (VMM) domain is tagged as a security domain, the users contained in the security domain can access the correspondingly tagged VMM domain. For example, if a tenant named solar is tagged with the security domain called sun and a VMM domain is also tagged with the security domain called sun, then users in the solar tenant can access the VMM domain according to their access rights.

AAA RBAC Roles and Privileges

The Application Policy Infrastructure Controller (APIC) provides the following AAA roles and privileges:

Role	Privilege	Description
aaa	aaa	Used for configuring authentication, authorization, accounting, and import/export policies.
admin	admin	Provides full access to all of the features of the fabric. The admin privilege can be considered to be a union of all other privileges.

Role: access-admin	
Privilege	Description
access-connectivity-l1	Used for Layer 1 configuration under infra. Example: selectors and port Layer 1 policy configurations.
access-connectivity-l2	Used for Layer 2 configuration under infra. Example: encapsulations on selectors, and attachable entity.
access-connectivity-l3	Used for Layer 3 configuration under infra and static route configurations under a tenant's L3Out.
access-connectivity-mgmt	Used for management infra policies.

Role: access-admin	
Privilege	Description
access-connectivity-util	Used for tenant ERSPAN policies.
access-equipment	Used for access port configuration.
access-protocol-l1	Used for Layer 1 protocol configurations under infra.
access-protocol-l2	Used for Layer 2 protocol configurations under infra.
access-protocol-l3	Used for Layer 3 protocol configurations under infra.
access-protocol-mgmt	Used for fabric-wide policies for NTP, SNMP, DNS, and image management.
access-protocol-ops	Used for operations-related access policies such as cluster policy and firmware policies.
access-qos	Used for changing CoPP and QoS-related policies.

Role:fabric-admin	
Privilege	Description
fabric-connectivity-l1	Used for Layer 1 configuration under the fabric. Example: selectors and port Layer 1 policy and vPC protection.
fabric-connectivity-l2	Used in firmware and deployment policies for raising warnings for estimating policy deployment impact.
fabric-connectivity-l3	Used for Layer 3 configuration under the fabric. Example: Fabric IPv4, IPv6, and MAC protection groups.
fabric-connectivity-mgmt	Used for atomic counter and diagnostic policies on leaf switches and spine switches.
fabric-connectivity-util	Used for atomic counter, diagnostic, and image management policies on leaf switches and spine switches.
fabric-equipment	Used for atomic counter, diagnostic, and image management policies on leaf switches and spine switches.
fabric-protocol-l1	Used for Layer 1 protocol configurations under the fabric.
fabric-protocol-l2	Used for Layer 2 protocol configurations under the fabric.
fabric-protocol-l3	Used for Layer 3 protocol configurations under the fabric.

Role:fabric-admin	
Privilege	Description
fabric-protocol-mgmt	Used for fabric-wide policies for NTP, SNMP, DNS, and image management.
fabric-protocol-ops	Used for ERSPAN and health score policies.
fabric-protocol-util	Used for firmware management traceroute and endpoint tracking policies.
tenant-connectivity-util	Used for atomic counter, diagnostic, and image management policies on leaf switches and spine switches.
tenant-connectivity-l2	Used for Layer 2 connectivity changes, including bridge domains and subnets.
tenant-connectivity-l3	Used for Layer 3 connectivity changes, including VRFs.
tenant-protocol-ops	Used for tenant traceroute policies.

Role	Privilege	Description
nw-svc-admin	nw-svc-device	Used for managing Layer 4 to Layer 7 service devices.
	nw-svc-devshare	Used for managing shared Layer 4 to Layer 7 service devices.
	nw-svc-policy	Used for managing Layer 4 to Layer 7 network service orchestration.
nw-svc-params	nw-svc-params	Used for managing Layer 4 to Layer 7 service policies.

Role: ops	
Privilege	Description
ops	Used for operational policies including monitoring and troubleshooting policies such as atomic counter, SPAN, TSW, tech support, traceroute, analytics, and core policies.

Role: read-all	
Privilege	Description
access-connectivity-l1	Used for Layer 1 configuration under infra. Example: selectors and port Layer 1 policy configurations.
access-connectivity-l2	Used for Layer 2 configuration under infra. Example: Encap configurations on selectors, and attachable entity.
access-connectivity-l3	Used for Layer 3 configuration under infra and static route configurations under a tenant's L3Out.
access-connectivity-mgmt	Used for management infra policies.
access-connectivity-util	Used for tenant ERSPAN policies.
access-equipment	Used for access port configuration.
access-protocol-l1	Used for Layer 1 protocol configurations under infra.
access-protocol-l2	Used for Layer 2 protocol configurations under infra.
access-protocol-l3	Used for Layer 3 protocol configurations under infra.
access-protocol-mgmt	Used for fabric-wide policies for NTP, SNMP, DNS, and image management.
access-protocol-ops	Used for operations-related access policies such as cluster policy and firmware policies.
access-qos	Used for changing CoPP and QoS-related policies.
fabric-connectivity-l1	Used for Layer 1 configuration under the fabric. Example: selectors and port Layer 1 policy and vPC protection.
fabric-connectivity-l2	Used in firmware and deployment policies for raising warnings for estimating policy deployment impact.
fabric-connectivity-l3	Used for Layer 3 configuration under the fabric. Example: Fabric IPv4, IPv6, and MAC protection groups.
fabric-protocol-l1	Used for Layer 1 protocol configurations under the fabric.
fabric-protocol-l2	Used for Layer 2 protocol configurations under the fabric.
fabric-protocol-l3	Used for Layer 3 protocol configurations under the fabric.
nw-svc-device	Used for managing Layer 4 to Layer 7 service devices.
nw-svc-devshare	Used for managing shared Layer 4 to Layer 7 service devices.

Role: read-all	
Privilege	Description
nw-svc-params	Used for managing Layer 4 to Layer 7 service policies.
nw-svc-policy	Used for managing Layer 4 to Layer 7 network service orchestration.
ops	Used for operational policies including monitoring and troubleshooting policies such as atomic counter, SPAN, TSW, tech support, traceroute, analytics, and core policies.
tenant-connectivity-util	Used for atomic counter, diagnostic, and image management policies on leaf switches and spine switches.
tenant-connectivity-l2	Used for Layer 2 connectivity changes, including bridge domains and subnets.
tenant-connectivity-l3	Used for Layer 3 connectivity changes, including VRFs.
tenant-connectivity-mgmt	Used for tenant in-band and out-of-band management connectivity configurations and for debugging/monitoring policies such as atomic counters and health score.
tenant-epg	Used for managing tenant configurations such as deleting/creating endpoint groups, VRFs, and bridge domains.
tenant-ext-connectivity-l1	Used for write access firmware policies.
tenant-ext-connectivity-l2	Used for managing tenant L2Out configurations.
tenant-ext-connectivity-l3	Used for managing tenant L3Out configurations.
tenant-ext-connectivity-mgmt	Used as write access for firmware policies.
tenant-ext-connectivity-util	Used for debugging/monitoring/observer policies such as traceroute, ping, oam, and eptrk.
tenant-ext-protocol-l1	Used for managing tenant external Layer 1 protocols. Generally only used for write access for firmware policies.
tenant-ext-protocol-l2	Used for managing tenant external Layer 2 protocols. Generally only used for write access for firmware policies.
tenant-ext-protocol-l3	Used for managing tenant external Layer 3 protocols such as BGP, OSPF, PIM, and IGMP.
tenant-ext-protocol-mgmt	Used as write access for firmware policies.
tenant-ext-protocol-util	Used for debugging/monitoring/observer policies such as traceroute, ping, oam, and eptrk.

Role: read-all	
Privilege	Description
tenant-network-profile	Used for managing tenant configurations, such as deleting and creating network profiles, and deleting and creating endpoint groups.
tenant-protocol-l1	Used for managing configurations for Layer 1 protocols under a tenant.
tenant-protocol-l2	Used for managing configurations for Layer 2 protocols under a tenant.
tenant-protocol-l3	Used for managing configurations for Layer 3 protocols under a tenant.
tenant-protocol-mgmt	Only used as write access for firmware policies.
tenant-protocol-ops	Used for tenant traceroute policies.
tenant-QoS	Used for QoS-related configurations for a tenant.
tenant-security	Used for contract-related configurations for a tenant.
vmm-connectivity	Used to read all the objects in APIC's VMM inventory required for virtual machine connectivity.
vmm-ep	Used to read virtual machine and hypervisor endpoints in the APIC's VMM inventory.
vmm-policy	Used for managing policies for virtual machine networking.
vmm-protocol-ops	Not used by VMM policies.
vmm-security	Used for managing authentication policies for VMM, such as the username and password for VMware vCenter.

Role: tenant-admin	
Privilege	Description
aaa	Used for configuring authentication, authorization, accounting and import/export policies.
access-connectivity-l1	Used for Layer 1 configuration under infra. Example: selectors and port Layer 1 policy configurations.
access-connectivity-l2	Used for Layer 2 configuration under infra. Example: Encap configurations on selectors, and attachable entity.
access-connectivity-l3	Used for Layer 3 configuration under infra and static route configurations under a tenant's L3Out.

Role: tenant-admin	
Privilege	Description
access-connectivity-mgmt	Used for management infra policies.
access-connectivity-util	Used for tenant ERSPAN policies.
access-equipment	Used for access port configuration.
access-protocol-l1	Used for Layer 1 protocol configurations under infra.
access-protocol-l2	Used for Layer 2 protocol configurations under infra.
access-protocol-l3	Used for Layer 3 protocol configurations under infra.
access-protocol-mgmt	Used for fabric-wide policies for NTP, SNMP, DNS, and image management.
access-protocol-ops	Used for operations-related access policies such as cluster policy and firmware policies.
access-qos	Used for changing CoPP and QoS-related policies.
fabric-connectivity-l1	Used for Layer 1 configuration under the fabric. Example: selectors and port Layer 1 policy and vPC protection.
fabric-connectivity-l2	Used in firmware and deployment policies for raising warnings for estimating policy deployment impact.
fabric-connectivity-l3	Used for Layer 3 configuration under the fabric. Example: Fabric IPv4, IPv6, and MAC protection groups.
fabric-connectivity-mgmt	Used for atomic counter and diagnostic policies on leaf switches and spine switches.
fabric-connectivity-util	Used for atomic counter, diagnostic, and image management policies on leaf switches and spine switches.
fabric-equipment	Used for atomic counter, diagnostic, and image management policies on leaf switches and spine switches.
fabric-protocol-l1	Used for Layer 1 protocol configurations under the fabric.
fabric-protocol-l2	Used for Layer 2 protocol configurations under the fabric.
fabric-protocol-l3	Used for Layer 3 protocol configurations under the fabric.
fabric-protocol-mgmt	Used for fabric-wide policies for NTP, SNMP, DNS, and image management.

Role: tenant-admin	
Privilege	Description
fabric-protocol-ops	Used for ERSPAN and health score policies.
fabric-protocol-util	Used for firmware management traceroute and endpoint tracking policies.
nw-svc-device	Used for managing Layer 4 to Layer 7 service devices.
nw-svc-devshare	Used for managing shared Layer 4 to Layer 7 service devices.
nw-svc-params	Used for managing Layer 4 to Layer 7 service policies.
nw-svc-policy	Used for managing Layer 4 to Layer 7 network service orchestration.
ops	Used for operational policies including monitoring and troubleshooting policies such as atomic counter, SPAN, TSW, tech support, traceroute, analytics, and core policies.
tenant-connectivity-util	Used for atomic counter, diagnostic, and image management policies on leaf switches and spine switches.
tenant-connectivity-l2	Used for Layer 2 connectivity changes, including bridge domains and subnets.
tenant-connectivity-l3	Used for Layer 3 connectivity changes, including VRFs.
tenant-connectivity-mgmt	Used for tenant in-band and out-of-band management connectivity configurations and for debugging/monitoring policies such as atomic counters and health score.
tenant-epg	Used for managing tenant configurations such as deleting/creating endpoint groups, VRFs, and bridge domains.
tenant-ext-connectivity-l1	Used for write access firmware policies.
tenant-ext-connectivity-l2	Used for managing tenant L2Out configurations.
tenant-ext-connectivity-l3	Used for managing tenant L3Out configurations.
tenant-ext-connectivity-mgmt	Used as write access for firmware policies.
tenant-ext-connectivity-util	Used for debugging/monitoring/observer policies such as traceroute, ping, oam, and eptrk.
tenant-ext-protocol-l1	Used for managing tenant external Layer 1 protocols. Generally only used for write access for firmware policies.

Role: tenant-admin	
Privilege	Description
tenant-ext-protocol-l2	Used for managing tenant external Layer 2 protocols. Generally only used for write access for firmware policies.
tenant-ext-protocol-l3	Used for managing tenant external Layer 3 protocols such as BGP, OSPF, PIM, and IGMP.
tenant-ext-protocol-mgmt	Used as Write access for firmware policies.
tenant-ext-protocol-util	Used for debugging/monitoring/observer policies such as traceroute, ping, oam, and eptrk.
tenant-network-profile	Used for managing tenant configurations, such as deleting and creating network profiles, and deleting and creating endpoint groups.
tenant-protocol-l1	Used for managing configurations for Layer 1 protocols under a tenant.
tenant-protocol-l2	Used for managing configurations for Layer 2 protocols under a tenant.
tenant-protocol-l3	Used for managing configurations for Layer 3 protocols under a tenant.
tenant-protocol-mgmt	Only used as write access for firmware policies.
tenant-protocol-ops	Used for tenant traceroute policies.
tenant-QoS	Used for QoS-related configurations for a tenant.
tenant-security	Used for contract-related configurations for a tenant.
vmm-connectivity	Used to read all the objects in APIC's VMM inventory required for virtual machine connectivity.
vmm-ep	Used to read virtual machine and hypervisor endpoints in the APIC's VMM inventory.
vmm-policy	Used for managing policies for virtual machine networking.
vmm-protocol-ops	Not used by VMM policies.
vmm-security	Used for managing authentication policies for VMM, such as the username and password for VMware vCenter.

Role: tenant-ext-admin	
Privilege	Description
tenant-connectivity-util	Used for atomic counter, diagnostic, and image management policies on leaf switches and spine switches.
tenant-connectivity-l2	Used for Layer 2 connectivity changes, including bridge domains and subnets.
tenant-connectivity-l3	Used for Layer 3 connectivity changes, including VRFs.
tenant-connectivity-mgmt	Used for tenant in-band and out-of-band management connectivity configurations and for debugging/monitoring policies such as atomic counters and health score.
tenant-epg	Used for managing tenant configurations such as deleting/creating endpoint groups, VRFs, and bridge domains.
tenant-ext-connectivity-l1	Used for write access firmware policies.
tenant-ext-connectivity-l2	Used for managing tenant L2Out configurations.
tenant-ext-connectivity-l3	Used for managing tenant L3Out configurations.
tenant-ext-connectivity-mgmt	Used as write access for firmware policies.
tenant-ext-connectivity-util	Used for debugging/monitoring/observer policies such as traceroute, ping, oam, and eptrk.
tenant-ext-protocol-l1	Used for managing tenant external Layer 1 protocols. Generally only used for write access for firmware policies.
tenant-ext-protocol-l2	Used for managing tenant external Layer 2 protocols. Generally only used for write access for firmware policies.
tenant-ext-protocol-l3	Used for managing tenant external Layer 3 protocols such as BGP, OSPF, PIM, and IGMP.
tenant-ext-protocol-mgmt	Used as Write access for firmware policies.
tenant-ext-protocol-util	Used for debugging/monitoring/observer policies such as traceroute, ping, oam, and eptrk.
tenant-network-profile	Used for managing tenant configurations, such as deleting and creating network profiles, and deleting and creating endpoint groups.
tenant-protocol-l1	Used for managing configurations for Layer 1 protocols under a tenant.
tenant-protocol-l2	Used for managing configurations for Layer 2 protocols under a tenant.

Role: tenant-ext-admin	
Privilege	Description
tenant-protocol-l3	Used for managing configurations for Layer 3 protocols under a tenant.
tenant-protocol-mgmt	Only used as write access for firmware policies.
tenant-protocol-ops	Used for tenant traceroute policies.
tenant-QoS	Used for QoS-related configurations for a tenant.
tenant-security	Used for contract-related configurations for a tenant.
vmm-connectivity	Used to read all the objects in APIC's VMM inventory required for virtual machine connectivity.
vmm-ep	Used to read virtual machine and hypervisor endpoints in the APIC's VMM inventory.
vmm-policy	Used for managing policies for virtual machine networking.
vmm-protocol-ops	Not used by VMM policies.
vmm-security	Used for managing authentication policies for VMM, such as the username and password for VMware vCenter.

Role: vmm-admin	
Privilege	Description
vmm-connectivity	Used to read all the objects in APIC's VMM inventory required for virtual machine connectivity.
vmm-ep	Used to read virtual machine and hypervisor endpoints in the APIC's VMM inventory.
vmm-policy	Used for managing policies for virtual machine networking.
vmm-protocol-ops	Not used by VMM policies.
vmm-security	Used for managing authentication policies for a VMM, such as the username and password for VMware vCenter.

Custom Roles

You can create custom roles and assign privileges to the roles. The interface internally assigns one or more privileges to all managed object classes. In an XML model, privileges are assigned in an access attribute. Privilege bits are assigned at compile time and apply per class, and not per instance or object of the class.

In addition to the 45 privilege bits, the "aaa" privilege bit applies to all AAA-subsystem configuration and read operations. The following table provides a matrix of the supported privilege combinations. The rows in the table represent Cisco Application Centric Infrastructure (ACI) modules and the columns represent functionality for a given module. A value of "Yes" in a cell indicates that the functionality for the module is accessible and there exists a privilege bit to access that functionality. An empty cell indicates that the particular functionality for module is not accessible by any privilege bit. See the privilege bit descriptions to learn what each bit does.

	Connectivity	QoS	Security	Application	Fault	Stats	Provider	Service Profile	Service Chain
VMM	Yes		Yes		Yes	Yes	Yes		
Fabric	Yes	Yes	Yes	Yes	Yes	Yes	Yes		
External	Yes	Yes	Yes		Yes	Yes			Yes
Tenant	Yes	Yes	Yes	EPG, NP	Yes	Yes			Yes
Infra	Yes	Yes	Yes	Yes	Yes	Yes			Yes
Ops					Yes	Yes			
Storage	Yes	Yes	Yes	Yes	Yes	Yes			
Network Service	Yes	Yes	Yes	Yes	Yes	Yes		Yes	

Sample RBAC Rules

The RBAC rules in the sample JSON file below enable both trans-tenant access and tenant access to a VMM domain resource. The resources needed by the consumer are `uni/tn-prov1/brc-webCtrct` and `vmmmp-Vmware/dom-Datacenter`.

The following two RBAC rules enable the consumer tenant to post the consumer postman query in the JSON file below.

```
<aaaRbacEp>
  <aaaRbacRule objectDn="uni/vmmmp-VMware/dom-Datacenter" domain="cons1"/>
  <aaaRbacRule objectDn="uni/tn-prov1/brc-webCtrct" domain="cons1"/>
</aaaRbacEp>
```

The JSON file below contains these two RBAC rules:

```
{"id": "ac62a200-9210-f53b-7114-a8f4cffb9a36", "name": "SharedContracts", "timestamp": 1398806919868, "requests": [{"collectionId": "ac62a200-9210-f53b-7114-a8f4cffb9a36", "id": "2dfc75cc-431e-e136-622c-a577ce7622d8", "name": "login as prov1", "description": ""},
```

```

"url":"http://http://solar.local:8000/api/aaaLogin.json",
"method":"POST",
"headers":"",
"data":
{"\\"aaaUser\\":{\\"attributes\\\":{\\"name\\\": \\"prov1\\\", \\"pwd\\\": \\"secret!\\\"}}}},
"dataType":"raw","timestamp":0,"version":2,"time":1398807562828},
{"collectionId":"ac62a200-9210-f53b-7114-a8f4cffb9a36","id":"56e46db0-77ea-743f-a64e-c5f7b1f59807",
"name":"Root login",
"description":"",
"url":"http://http://solar.local:8000/api/aaaLogin.json",
"method":"POST",
"headers":"",
"data":
{"\\"aaaUser\\":{\\"attributes\\\":{\\"name\\\": \\"admin\\\", \\"pwd\\\": \\"secret!\\\"}}}},
"dataType":"raw","timestamp":0,"responses":[],"version":2},
 {"collectionId":"ac62a200-9210-f53b-7114-a8f4cffb9a36","id":"804893f1-0915-6d35-169d-3af0eb3e64ec",
"name":"consumer tenant only",
"description":"",
"url":"http://http://solar.local:8000/api/policymgr/mo/uni/tn-cons1.xml",
"method":"POST",
"headers":"",
"data":
"<fvTenant name=\"cons1\">
    <aaaDomainRef name=\"cons1\"/>\n</fvTenant>\n",
"dataType":"raw","timestamp":0,"version":2,"time":1398968007487},
 {"collectionId":"ac62a200-9210-f53b-7114-a8f4cffb9a36","id":"85802d50-8089-bf8b-4481-f149bec258c8",
"name":"login as cons1",
"description":"",
"url":"http://solar.local:8000/api/aaaLogin.json",
"method":"POST",
"headers":"",
"data":
{"\\"aaaUser\\":{\\"attributes\\\":{\\"name\\\": \\"cons1\\\", \\"pwd\\\": \\"secret!\\\"}}}},
"dataType":"raw","timestamp":0,"version":2,"time":1398807575531},
 {"collectionId":"ac62a200-9210-f53b-7114-a8f4cffb9a36","id":"a2739d92-5f9d-f16c-8894-0f64b6f967a3",
"name":"consumer",
"description":"",
"url":"http://solar.local:8000/api/policymgr/mo/uni/tn-cons1.xml",
"method":"POST","headers":"","data":
"<fvTenant name=\"cons1\" status=\"modified\">\n    <fvCtx name=\"cons1\"/>\n        <!-- bridge domain -->\n            <fvBD name=\"cons1\"/>\n                <fvRsCtx tnFvCtxName=\"cons1\" />\n                    <fvSubnet ip=\"10.0.2.128/24\" scope='shared' />\n                </fvBD>\n            <!-- DNS Shared Service Contract Interface-->\n                <vzCPIf name=\"consIf\"/>\n                    <vzRsIf tDn=\"uni/tn-prov1/brc-webCtrct\" />\n                </vzRsIf>\n            </vzCPIf>\n        </fvCtx>\n    <fvAp name=\"cons1\"/>\n        <fvAEPg name=\"APP\"/>\n            <fvRsBd tnFvBDName=\"cons1\" />\n                <fvRsNodeAtt tDn=\"topology/pod-1/node-101\" encaps=\"vlan-4000\" instrImedcy=\"immediate\" mode=\"regular\"/>\n                    <fvRsDomAtt tDn=\"uni/vmmp-VMware/dom-Datacenter\"/>\n                    <fvRsConsIf tnVzCPIfName=\"consIf\"/>\n            </fvAEPg>\n        </fvAp>\n    </fvTenant>\n",
"dataType":"raw","timestamp":0,"version":2,"time":1398818639692},
 {"collectionId":"ac62a200-9210-f53b-7114-a8f4cffb9a36","id":"c0bd866d-600a-4f45-46ec-6986398cbf78",
"name":"provider tenant only",
"description":"",
"url":"http://solar.local:8000/api/policymgr/mo/uni/tn-prov1.xml",
"method":"POST",

```

```

"headers":"",
"data":
"<fvTenant name=\"prov1\"><aaaDomainRef name=\"prov1\"/>
 \n
</fvTenant>\n",
"dataMode":"raw","timestamp":0,"version":2,"time":1398818137518},
{"collectionId":"ac62a200-9210-f53b-7114-a8f4cffb9a36","id":"d433a213-e95d-646d-895e-3a9e2e2b7ba3",
"name":"create RbacRule",
"description":"",
"url":"http://solar.local:8000/api/policymgr/mo/uni.xml",
"method":"POST",
"headers":"",
"data":
"<aaaRbacEp>\n
  <aaaRbacRule objectDn=\"uni/vmmp-VMware/dom-Datacenter\" domain=\"cons1\"/>\n
  <aaaRbacRule objectDn=\"uni/tn-prov1/brc-webCtrct\" domain=\"cons1\"/>\n
</aaaRbacEp>\n",
"dataMode":"raw","timestamp":0,"version":2,"time":1414195420515},
{"collectionId":"ac62a200-9210-f53b-7114-a8f4cffb9a36","id":"d5c5d580-a11a-7c61-34ac-cbdac249157f",
"name":"provider",
"description":"",
"url":"http://solar.local:8000/api/policymgr/mo/uni/tn-prov1.xml",
"method":"POST",
"headers":"",
"data":
"<fvTenant name=\"prov1\" status=\"modified\">\n
  <fvCtx name=\"prov1\"/>\n
  \n <!-- bridge domain -->\n
    <fvBD name=\"prov1\">\n
      <fvRsCtx tnFvCtxName=\"prov1\" />\n
    </fvBD>\n \n
    <vzFilter name='t0f0' >\n
      <vzEntry etherT='ip' dToPort='10' prot='6' name='t0f0e9' dFromPort='10'>
      </vzEntry>\n
    </vzFilter>\n \n
    <vzFilter name='t0f1'>\n
      <vzEntry etherT='ip' dToPort='209' prot='6' name='t0f1e8' dFromPort='109'>
      </vzEntry>\n
    </vzFilter>\n \n
    <vzBrCP name=\"webCtrct\" scope=\"global\">\n
      <vzSubj name=\"app\">\n
        <vzRsSubjFiltAtt tnVzFilterName=\"t0f0\"/>\n
      <vzRsSubjFiltAtt tnVzFilterName=\"t0f1\"/>\n
    </vzSubj>\n
  </vzBrCP>\n \n
  <fvAp name=\"prov1AP\">\n
    <fvAEPg name=\"Web\">\n
      <fvRsBd tnFvBDName=\"prov1\" />\n
        <fvRsNodeAtt tDn=\"topology/pod-1/node-17\" encap=\"vlan-4000\"/>
      instrImedcy=\"immediate\" mode=\"regular\"/>\n
      <fvRsProv tnVzBrCPName=\"webCtrct\"/>\n
    <fvRsDomAtt tDn=\"uni/vmmp-VMware/dom-Datacenter\"/>\n
      <fvSubnet ip=\"10.0.1.128/24\" scope='shared'/'>\n
    </fvAEPg>\n
  </fvAp>\n
</fvTenant>\n",
"dataMode":"raw","timestamp":0,"version":2,"time":1398818660457},
{"collectionId":"ac62a200-9210-f53b-7114-a8f4cffb9a36","id":"e8866493-2188-8893-8e0c-4ca0903b18b8",
"name":"add user prov1",
"description":"",
"url":"http://solar.local:8000/api/policymgr/mo/uni/userext.xml",
"method":"POST",
"headers":"",
"data":
"<aaaUserEp>\n
  <aaaUser name=\"prov1\" pwd=\"secret!\">
    <aaaUserDomain name=\"prov1\">
      <aaaUserRole name=\"tenant-admin\" privType=\"writePriv\"/>
      <aaaUserRole name=\"vmm-admin\" privType=\"writePriv\"/>
    </aaaUserDomain>
  </aaaUser>
</aaaUserEp>\n"
}

```

```

</aaaUser>\n
    <aaaUser name=\"cons1\" pwd=\"secret!\">
    <aaaUserDomain name=\"cons1\">
        <aaaUserRole name=\"tenant-admin\" privType=\"writePriv\"/>
        <aaaUserRole name=\"vmm-admin\" privType=\"writePriv\"/>
    </aaaUserDomain>
</aaaUser>\n
    <aaaDomain name=\"prov1\"/>\n
    <aaaDomain name=\"cons1\"/>\n
</aaaUserEp>\n",
"dataMode":"raw","timestamp":0,"version":2,"time":1398820966635}]}

```

Enabling Port Security

About Port Security and ACI

The port security feature protects the ACI fabric from being flooded with unknown MAC addresses by limiting the number of MAC addresses learned per port. The port security feature support is available for physical ports, port channels, and virtual port channels.

Port Security Guidelines and Restrictions

The guidelines and restrictions are as follows:

- Port security is available per port.
- Port security is supported for physical ports, port channels, and virtual port channels (vPCs).
- Static and dynamic MAC addresses are supported.
- MAC address moves are supported from secured to unsecured ports and from unsecured ports to secured ports.
- The MAC address limit is enforced only on the MAC address and is not enforced on a MAC and IP address.
- Port security is not supported with the Fabric Extender (FEX).

Port Security and Learning Behavior

For non-vPC ports or port channels, whenever a learn event comes for a new endpoint, a verification is made to see if a new learn is allowed. If the corresponding interface has a port security policy not configured or disabled, the endpoint learning behavior is unchanged with what is supported. If the policy is enabled and the limit is reached, the current supported action is as follows:

- Learn the endpoint and install it in the hardware with a drop action.
- Silently discard the learn.

If the limit is not reached, the endpoint is learned and a verification is made to see if the limit is reached because of this new endpoint. If the limit is reached, and the learn disable action is configured, learning will be disabled in the hardware on that interface (on the physical interface or on a port channel or vPC). If the limit is reached and the learn disable action is not configured, the endpoint will be installed in hardware with a drop action. Such endpoints are aged normally like any other endpoints.

When the limit is reached for the first time, the operational state of the port security policy object is updated to reflect it. A static rule is defined to raise a fault so that the user is alerted. A syslog is also raised when the limit is reached.

In case of vPC, when the MAC limit is reached, the peer leaf switch is also notified so learning can be disabled on the peer. As the vPC peer can be rebooted any time or vPC legs can become unoperational or restart, this state will be reconciled with the peer so vPC peers do not go out of sync with this state. If they get out of sync, there can be a situation where learning is enabled on one leg and disabled on the other leg.

By default, once the limit is reached and learning is disabled, it will not be automatically re-enabled and there is no automatic recovery policy. As this is a security violation, the administrator must rectify the problem and toggle the action to clear the hardware state so that learning is enabled again.

Port Security at Port Level

In the APIC, the user can configure the port security on switch ports. Once the MAC limit has exceeded the maximum configured value on a port, all traffic from the exceeded MAC addresses is forwarded. The following attributes are supported:

- **Violation**—The violation action is available in protect mode. In the protect mode, MAC learning is disabled and MAC addresses are not added to the CAM table. To re-enable MAC learning after a violation, the port security policy must be disabled and enabled once again.
- **Maximum**—The current supported range for the maximum endpoints configured value is 0-12000. If the maximum endpoints value is 0, the port security policy is disabled on that port.

Protect Mode

The protect mode prevents further port security violations from occurring. Once the MAC limit exceeds the maximum configured value on a port, all traffic from excess MAC addresses will be dropped and further learning is disabled.

Configuring Port Security Using REST API

Procedure

Configure the port security.

Example:

```
<polUni>
<infraInfra>

<l2PortSecurityPol name="testL2PortSecurityPol" maximum="10" violation="protect"/>

<infraNodeP name="test">
    <infraLeafS name="test" type="range">
        <infraNodeBlk name="test" from_="101" to_="102"/>
    </infraLeafS>
    <infraRsAccPortP tDn="uni/infra/accportprof-test"/>
</infraNodeP>

    <infraAccPortP name="test">
        <infraHPortS name="pse1c" type="range">
            <infraPortBlk name="blk"
```

```

        fromCard="1" toCard="1" fromPort="20" toPort="22">
    </infraPortBlk>
<infraRsAccBaseGrp tDn="uni/infra/funcprof/accportgrp-testPortG" />
</infraHPorts>
</infraAccPortP>

<infraFuncP>
<infraAccPortGrp name="testPortG">
    <infraRsL2PortSecurityPol tnL2PortSecurityPolName="testL2PortSecurityPol"/>

    <infraRsAttEntP tDn="uni/infra/attentp-test" />
</infraAccPortGrp>
</infraFuncP>

<infraAttEntityP name="test">
    <infraRsDomP tDn="uni/phys-mininet"/>
</infraAttEntityP>
</infraInfra>
</polUni>

```

Enabling COOP Authentication

Overview

Council of Oracle Protocol (COOP) is used to communicate the mapping information (location and identity) to the spine proxy. A leaf switch forwards endpoint address information to the spine switch 'Oracle' using Zero Message Queue (ZMQ). COOP running on the spine nodes will ensure all spine nodes maintain a consistent copy of endpoint address and location information and additionally maintain the distributed hash table (DHT) repository of endpoint identity to location mapping database.

COOP data path communication provides high priority to transport using secured connections. COOP is enhanced to leverage the MD5 option to protect COOP messages from malicious traffic injection. The APIC controller and switches support COOP protocol authentication.

COOP protocol is enhanced to support two ZMQ authentication modes: strict and compatible.

- Strict mode: COOP allows MD5 authenticated ZMQ connections only.
- Compatible mode: COOP accepts both MD5 authenticated and non-authenticated ZMQ connections for message transportation.

Using COOP with Cisco APIC

To support COOP Zero Message Queue (ZMQ) authentication support across the Cisco Application Centric Infrastructure (ACI) fabric, the Application Policy Infrastructure Controller (APIC) supports the MD5 password and also supports the COOP secure mode.

COOP ZMQ Authentication Type Configuration—A new managed object, `coop:AuthP`, is added to the Data Management Engine (DME)/COOP database (`coop/inst/auth`). The default value for the attribute type is "compatible", and users have the option to configure the type to be "strict".

COOP ZMQ Authentication MD5 password—The APIC provides a managed object (`fabric:SecurityToken`), that includes an attribute to be used for the MD5 password. An attribute in this managed object, called "token", is a string that changes every hour. COOP obtains the notification from the DME to update the password for ZMQ authentication. The attribute token value is not displayed.

Guidelines and Limitations

Follow these guidelines and limitations:

- During an ACI fabric upgrade, the COOP strict mode is disallowed until all switches are upgraded to a secure-enabled image. This protection prevents the unexpected rejection of a COOP connection that could be triggered by prematurely enabling the strict mode.
- During the upgrade or downgrade of APIC controllers, the ACI fabric may be in a state with mixed secure and non-secure APIC images. If the authentication mode on the secure image is strict, some switches may remain in an out of service state during the upgrade or downgrade process. If the authentication mode type is compatible mode, the upgrade will not cause the switches to be out-of-service.

Configuring COOP Authentication Using the REST API

Procedure

Configure a COOP authentication policy.

In the example, the strict mode is chosen.

Example:

`https://172.23.53.xx/api/node/mo/uni/fabric/pol-default.xml`

```
<coopPol type="strict">  
</coopPol>
```

