How to configure Overlay Fabric with OpenClos

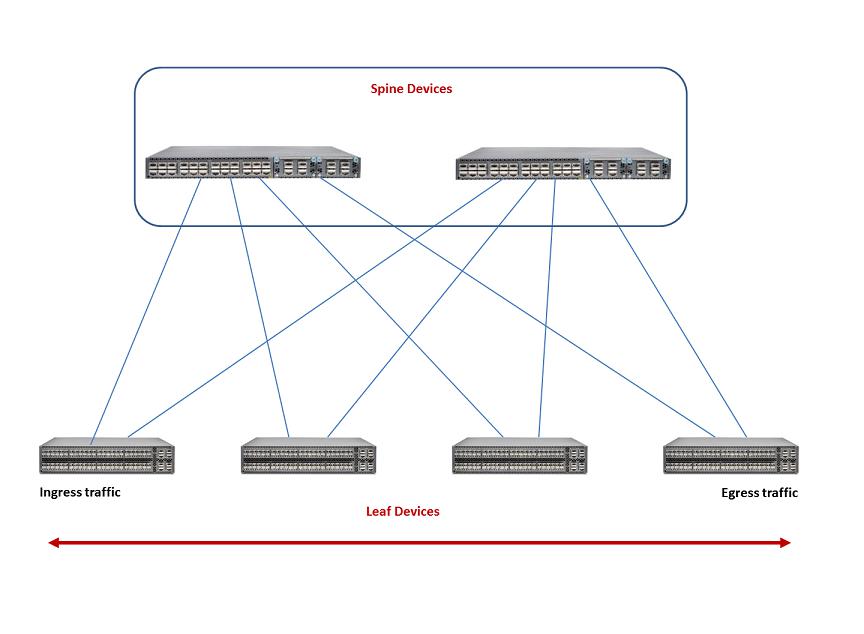
Understanding Layer 3 Fabrics



Most enterprises that host data centers are looking to increase resiliency and also support new technologies such as VMware NSX that allow them to deploy applications, servers, and virtual networks within seconds. Layer 3 Fabrics allow them to support better uptime, performance, and newer cloud infrastructures such as VMware NSX. In order to maintain the large scale required to host thousands of servers, the use of a multi-stage Clos architecture is required. Such an architecture allows the physical network to scale beyond the port density of a single switch. Layer 3 Fabrics use BGP as the control plane protocol to advertise prefixes, perform traffic engineering, and tag traffic. The most common designs in a multi-stage Clos architecture are a 3-stage and 5-stage networks that use the spine-and-leaf topology.

Spine-and-leaf topology is an alternate to the traditional three-layer network architecture, which consists of an access layer, aggregation layer, and a core. In the spine-and-leaf topology, all the leaf devices are connected to the spine devices in a mesh as shown in [Figure 1](#page600).

Figure 1: Layer 3 Fabric in a Spine and Leaf Topology



Typically, the spine devices are high-performance switches capable of Layer 3 switching and routing combined with high port density. Spine devices constitute the core and the leaf devices constitute the access layer in Layer 3 Fabrics. Leaf devices enable servers to connect to the Layer 3 Fabric. They also provide uplinks to spine devices.

OpenClos currently supports only the 3-stage design. The 3-stage design has two roles—the spine and the leaf. It is called a 3-stage design because the traffic must traverse three switches in the worst-case scenario.

The maximum number of spine devices that you can have in your Layer 3 Fabric depends on the number of 40-Gigabit Ethernet interfaces in your leaf devices. A Layer 3 Fabric that has 8 QFX5100-24Q spine devices and 32 QFX5100-96S leaf devices (each leaf supports 96 10-Gigabit Ethernet ports) can provide 3072 usable 10-Gigabit Ethernet ports.

VXLAN—EVPN Overlay Overview



Spanning Tree Protocol (STP), multichassis link aggregation group (MC-LAG), and Transparent Interconnection of Lots of Links (TRILL) were some of the commonly used technologies in traditional data centers. However, as the data centers started to grow exponentially, these technologies were not able to scale to meet the requirements of data centers. To cater to these requirements, data center administrators started using orchestration tools such as VMware vCenter, VMware vCenter with NSX, OpenStack, and OpenStack with NSX plug-in to orchestrate the networking needs of the tenants that a data center serves. This approach meant that the data center might require additional plug-ins to configure VLANs and gateways or that you might need to make changes to the physical network topology, to accommodate a new tenant. Most of the current day data centers handle multiple customer groups, organizations, or tenants that require a new data center architecture that decouples the underlay network from tenant overlay networks. A Layer 3 Fabric underlay coupled with a Virtual Extensible LAN (VXLAN)–Ethernet VPN (EVPN) overlay solution that uses bare-metal servers and/or virtual servers, or both Network Director for management enables data center and cloud operators to deploy much larger networks than that are otherwise possible with traditional Layer 2 Ethernet-based architectures.

Some of the major advantages that VXLAN—EVPN overlay networks provide are:

1. Scalability—Most enterprises accommodate their growth by increasing the use of cloud services, while others choose to deploy their own private and hybrid clouds. Service providers also must be able to be grow rapidly to have sufficient capacity to meet the demands of the enterprises. Today’s networks are often too rigid and difficult to change for scaling to meet the needs of large enterprises and service providers. But by using

VXLAN—EVPN, data centers and cloud operators can have up to 16 million overlay networks in a cloud data center.

1. Operational efficiency—As enterprises expand geographically, the physical distance between the data centers and users also increases, which makes timely maintenance and application mobility a challenge. The VXLAN—EVPN solution enables network administrators to easily migrate applications within the data center and between data centers for business continuity so that they can maintain the data center without downtime, for effective load balancing.
2. High Performance—End users often experience poor response times and even outages of business-critical applications caused by bandwidth limitations and latency problems. Multi-pathing and control plane learning features, that are part of the VXLAN—EVPN solution, can optimize network traffic flows, rein in network faults, and ensure maximum utilization of bandwidth.

Creating Overlay Fabrics

OpenClos builds an overlay fabric on top of a working IP fabric (underlay). Refer to “How to configure IP Fabric with OpenClos” for details. The following steps are based on assumption that you have configured IP fabric. You create Overlay Fabric by performing following tasks in the order listed here:

Start OpenClos

1. Start REST Server

OpenClos REST server supports both HTTP and HTTPS.

**HTTP**:

Configure HTTP with following in <OPENCLOS\_INSTALLATION\_PATH>/jnpr/openclos/conf/openclos.yaml,

restServer :

version : 1

protocol : http

ipAddr : <REST server ip>

port : <REST server port>

Use HTTP when your client is located in the same machine as the OpenClos REST server. In this case, no authentication is performed. The only configurations required in this case are 'ipAddr' and 'port'.

**HTTPS**:

Configure HTTPS with following in <OPENCLOS\_INSTALLATION\_PATH>/jnpr/openclos/conf/openclos.yaml,

restServer :

version : 1

protocol : https

ipAddr : <REST server ip>

port : <REST server port>

username : <username>

password : <2-way encrypted password>

certificate : <full path of the server certificate>

Use HTTPS when your client is accessing OpenClos REST server remotely. In this case, Basic Authentication is done over HTTPS.

***Default username/password/server certificate***:

OpenClos comes with a built-in username/password 'juniper/juniper' in openclos.yaml. OpenClos automatically generates a default server certificate whose CN subject is set to 'ipAddr' value in openclos.yaml when you start the REST server for the first time. The default server certificate is stored in ~/openclos.pem. The default username/password/server certificate enables user to use OpenClos HTTPS REST server out of box. The only configruations required in this case are 'ipAddr' and 'port'.

***Non default username/password/server certificate***:

If user decides to use non default username/password/server certificate, follow below instructions:

* Change 'username' to the new username
* Use "python crypt.py " to generate a 2-way encrypted password:
* root@sw-ubuntu25:# cd <OPENCLOS\_INSTALLATION\_PATH>/jnpr/openclos

root@sw-ubuntu25:.../jnpr/openclos# python crypt.py foobar

$9$lusvWxZGi5QnVwYoZG.m

Then copy-paste the output ($9$lusvWxZGi5QnVwYoZG.m) to 'password' in openclos.yaml

* Change 'certificate' to the full path of your certificate. Make sure your cerfiicate's CN subject is set to 'ipAddr' value in openclos.yaml
* Note the 'username', 'password' and 'cerficate' are required in HTTPS mode. OpenClos REST server won't start without them.
* Note the 'ipAddr' value needs to be set properly. OpenClos REST server won't start if 'ipAddr' value is 0.0.0.0
* Note Openclos also generates a public key file ~/openclos.pem.cer for the default certificate. User might want to import this file to their client if their client does not automatically accept the server cert.

Now Start REST Server

* root@sw-ubuntu25:# cd <OPENCLOS\_INSTALLATION\_PATH>/jnpr/openclos

root@sw-ubuntu25:.../jnpr/openclos# python rest.py

Create Overlay Device

Send HTTP POST request with following request body.

POST http://<server>:<port>/openclos/v1/overlay/devices

{

"device": {

"name": "leaf-0",

"role": "leaf",

"address": "10.92.82.12",

"routerId": "10.92.82.12",

"podName": "pod1",

"username": "root",

"password": "Embe1mpls"

}

}

HTTP Response from OpenClos

200 OK

{

"device": {

"description": null,

"podName": "pod1",

"uri": "http://192.168.56.101:20080/openclos/v1/overlay/devices/1600a0b0-9e15-4dd1-8bd6-80d6307a4e1c",

"fabrics": [],

"routerId": "10.92.82.12",

"role": "leaf",

"address": "10.92.82.12",

"id": "1600a0b0-9e15-4dd1-8bd6-80d6307a4e1c",

"name": "leaf-0"

}

}

Create Overlay Fabric

Send HTTP POST request with following request body.

POST http://<server>:<port>/openclos/v1/overlay/fabrics

{

"fabric": {

"name": "f1",

"routeReflectorAddress": "1.1.1.1",

"overlayAsn": "65001",

"devices": [

"1600a0b0-9e15-4dd1-8bd6-80d6307a4e1c"

]

}

}

HTTP Response from OpenClos

200 OK

{

"fabric": {

"description": null,

"uri": "http://192.168.56.101:20080/openclos/v1/overlay/fabrics/3873df0b-90ee-4e50-8487-5461007c3db9",

"devices": [

"http://192.168.56.101:20080/openclos/v1/overlay/devices/1600a0b0-9e15-4dd1-8bd6-80d6307a4e1c"

],

"overlayAsn": 65001,

"routeReflectorAddress": "1.1.1.1",

"tenants": [],

"id": "3873df0b-90ee-4e50-8487-5461007c3db9",

"name": "f1"

}

}

Create Overlay Tenant

Send HTTP POST request with following request body.

POST http://<server>:<port>/openclos/v1/overlay/tenants

{

"tenant": {

"name": "t1",

"fabric": "3873df0b-90ee-4e50-8487-5461007c3db9"

}

}

HTTP Response from OpenClos

200 OK

{

"tenant": {

"fabric": "http://192.168.56.101:20080/openclos/v1/overlay/fabrics/3873df0b-90ee-4e50-8487-5461007c3db9",

"uri": "http://192.168.56.101:20080/openclos/v1/overlay/tenants/933414ba-5446-469b-9b4f-87ec8560a865",

"name": "t1",

"vrfs": [],

"id": "933414ba-5446-469b-9b4f-87ec8560a865",

"description": null

}

}

Create Overlay Vrf

Send HTTP POST request with following request body.

POST http://<server>:<port>/openclos/v1/overlay/vrfs

{

"tenant": {

"name": "t1",

"fabric": "3873df0b-90ee-4e50-8487-5461007c3db9"

}

}

HTTP Response from OpenClos

200 OK

{

"vrf": {

"description": null,

"uri": "http://192.168.56.101:20080/openclos/v1/overlay/vrfs/ec496acd-f10b-44f2-a7e2-de2b040b7094",

"networks": [],

"loopbackAddress": "10.92.80.0/24",

"routedVnid": null,

"id": "ec496acd-f10b-44f2-a7e2-de2b040b7094",

"tenant": "http://192.168.56.101:20080/openclos/v1/overlay/tenants/933414ba-5446-469b-9b4f-87ec8560a865",

"name": "v2"

}

}

Create Overlay Network

Send HTTP POST request with following request body.

POST http://<server>:<port>/openclos/v1/overlay/networks

{

"network": {

"name": "n1",

"vlanid": "1000",

"vnid": "100",

"vrf": "ec496acd-f10b-44f2-a7e2-de2b040b7094"

}

}

HTTP Response from OpenClos

200 OK

{

"network": {

"pureL3Int": false,

"subnets": [],

"description": null,

"vnid": 100,

"l2ports": [],

"uri": "http://192.168.56.101:20080/openclos/v1/overlay/networks/d92d8263-2444-4c5d-82bc-fb28eedc17f6",

"vlanid": 1000,

"vrf": "http://192.168.56.101:20080/openclos/v1/overlay/vrfs/ec496acd-f10b-44f2-a7e2-de2b040b7094",

"aggregatedL2ports": [],

"id": "d92d8263-2444-4c5d-82bc-fb28eedc17f6",

"name": "n1"

}

}

Create Overlay Subnet

Send HTTP POST request with following request body.

POST http://<server>:<port>/openclos/v1/overlay/subnets

{

"subnet": {

"name": "s1",

"cidr": "10.92.1.0/24",

"network": "d92d8263-2444-4c5d-82bc-fb28eedc17f6"

}

}

HTTP Response from OpenClos

200 OK

{

"subnet": {

"l3ports": [],

"network": "http://192.168.56.101:20080/openclos/v1/overlay/networks/d92d8263-2444-4c5d-82bc-fb28eedc17f6",

"uri": "http://192.168.56.101:20080/openclos/v1/overlay/subnets/f2f7d56f-5b9b-4b7c-a481-c5f601e48cbb",

"name": "s1",

"cidr": "10.92.1.0/24",

"id": "f2f7d56f-5b9b-4b7c-a481-c5f601e48cbb",

"description": null

}

}

Create Overlay L2port

Send HTTP POST request with following request body.

POST http://<server>:<port>/openclos/v1/overlay/l2ports

{

"l2port": {

"name": "l2port1",

"interface": "xe-0/0/1",

"networks": [

"d92d8263-2444-4c5d-82bc-fb28eedc17f6"

],

"device": "1600a0b0-9e15-4dd1-8bd6-80d6307a4e1c"

}

}

HTTP Response from OpenClos

200 OK

{

"l2port": {

"description": null,

"uri": "http://192.168.56.101:20080/openclos/v1/overlay/l2ports/e8b7713c-7983-43d5-b549-a082a801e479",

"id": "e8b7713c-7983-43d5-b549-a082a801e479",

"device": "http://192.168.56.101:20080/openclos/v1/overlay/devices/1600a0b0-9e15-4dd1-8bd6-80d6307a4e1c",

"interface": "xe-0/0/1",

"networks": [

"http://192.168.56.101:20080/openclos/v1/overlay/networks/d92d8263-2444-4c5d-82bc-fb28eedc17f6"

],

"name": "l2port1"

}

}

Create Overlay Aggregated L2port (LAG)

Send HTTP POST request with following request body.

POST http://<server>:<port>/openclos/v1/overlay/aggregatedL2ports

{

"aggregatedL2port":{

"name":"ae0",

"esi":"00:80:e2:1c:d4:e7:d7:f8:ab:df",

"lacp":"a9:71:d2:d9:4e:bf",

"networks":[

"d92d8263-2444-4c5d-82bc-fb28eedc17f6"

],

"members":[

{

"device":"1600a0b0-9e15-4dd1-8bd6-80d6307a4e1c",

"interface":"xe-0/0/5"

},

{

"device":"d178f27f-38dc-461c-b498-ed31acb8285c",

"interface":"xe-0/0/5"

}

]

}

}

HTTP Response from OpenClos

200 OK

{

"aggregatedL2port": {

"description": null,

"uri": "http://192.168.56.101:20080/openclos/v1/overlay/aggregatedL2ports/26a90c73-1359-4eb9-8b27-84e45d5dff9a",

"networks": [

"http://192.168.56.101:20080/openclos/v1/overlay/networks/d92d8263-2444-4c5d-82bc-fb28eedc17f6"

],

"lacp": "a9:71:d2:d9:4e:bf",

"members": [

{

"device": "http://192.168.56.101:20080/openclos/v1/overlay/devices/1600a0b0-9e15-4dd1-8bd6-80d6307a4e1c",

"interface": "xe-0/0/5"

},

{

"device": "http://192.168.56.101:20080/openclos/v1/overlay/devices/d178f27f-38dc-461c-b498-ed31acb8285c",

"interface": "xe-0/0/5"

}

],

"id": "26a90c73-1359-4eb9-8b27-84e45d5dff9a",

"esi": "00:80:e2:1c:d4:e7:d7:f8:ab:df",

"name": "ae0"

}

}

Check Overlay Configuration Commit Status

* Check “all”

Send HTTP GET request with empty request body.

GET http://<server>:<port>/openclos/v1/overlay/deployStatus?id=<id>&scope=<scope>&object=<object>

id - object id created via previous steps

scope - all hierarchy of the object including all its ancestors, itself and all its descendants

- self hierarchy of the object including all its ancestors, itself but not including its descendants

object - type of the object:

fabric

tenant

vrf

network

subnet

l2port

aggregatedL2port

GET http://<server>:<port>/openclos/v1/overlay/deployStatus?object=fabric&scope=all&id=3873df0b-90ee-4e50-8487-5461007c3db9

HTTP Response from OpenClos

200 OK

{

"deployStatus": {

"fabrics": [

{

"deployDetail": [

{

"device": "leaf-0",

"status": "success",

"reason": null,

"configlet": "<omitted>"

}

],

"tenants": [

{

"deployDetail": [],

"uri": "http://192.168.56.101:20080/openclos/v1/overlay/tenants/933414ba-5446-469b-9b4f-87ec8560a865",

"vrfs": [

{

"deployDetail": [],

"uri": "http://192.168.56.101:20080/openclos/v1/overlay/vrfs/ec496acd-f10b-44f2-a7e2-de2b040b7094",

"networks": [

{

"l2ports": [

{

"deployDetail": [

{

"device": "leaf-0",

"status": "success",

"reason": null,

"configlet": "<omitted>"

}

],

"uri": "http://192.168.56.101:20080/openclos/v1/overlay/l2ports/e8b7713c-7983-43d5-b549-a082a801e479"

}

],

"deployDetail": [

{

"device": "leaf-0",

"status": "success",

"reason": null,

"configlet": "<omitted>"

}

],

"uri": "http://192.168.56.101:20080/openclos/v1/overlay/networks/d92d8263-2444-4c5d-82bc-fb28eedc17f6",

"aggregatedL2ports": [

{

"deployDetail": [

{

"device": "leaf-0",

"status": "success",

"reason": null,

"configlet": "<omitted>"

}

],

"uri": "http://192.168.56.101:20080/openclos/v1/overlay/aggregatedL2ports/26a90c73-1359-4eb9-8b27-84e45d5dff9a"

}

],

"subnets": [

{

"deployDetail": [],

"uri": "http://192.168.56.101:20080/openclos/v1/overlay/subnets/f2f7d56f-5b9b-4b7c-a481-c5f601e48cbb"

}

]

}

]

}

]

}

],

"uri": "http://192.168.56.101:20080/openclos/v1/overlay/fabrics/3873df0b-90ee-4e50-8487-5461007c3db9"

}

]

}

}