Software-Defined Network

Proxmox Server Solutions GmbH
<support@proxmox.com>

version 8.2.2, Thu Apr 25 09:24:16 CEST 2024

↩Index

Table of Contents

The **S**oftware-**D**efined **N**etwork (SDN) feature in Proxmox VE enables the creation of virtual zones and networks (VNets). This functionality simplifies advanced networking configurations and multitenancy setup.

Introduction

The Proxmox VE SDN allows for separation and fine-grained control of virtual guest networks, using flexible, software-controlled configurations.

Separation is managed through **zones**, virtual networks (**VNets**), and **subnets**. A zone is its own virtually separated network area. A VNet is a virtual network that belongs to a zone. A subnet is an IP range inside a VNet.

Depending on the type of the zone, the network behaves differently and offers specific features, advantages, and limitations.

Use cases for SDN range from an isolated private network on each individual node to complex overlay networks across multiple PVE clusters on different locations.

After configuring an VNet in the cluster-wide datacenter SDN administration interface, it is available as a common Linux bridge, locally on each node, to be assigned to VMs and Containers.

Support Status

History

The Proxmox VE SDN stack has been available as an experimental feature since 2019 and has been continuously improved and tested by many developers and users. With its integration into the web interface in Proxmox VE 6.2, a significant milestone towards broader integration was achieved. During the Proxmox VE 7 release cycle, numerous improvements and features were added. Based on user feedback, it became apparent that the fundamental design choices and their implementation were quite sound and stable. Consequently, labeling it as ‘experimental’ did not do justice to the state of the SDN stack. For Proxmox VE 8, a decision was made to lay the groundwork for full integration of the SDN feature by elevating the management of networks and interfaces to a core component in the Proxmox VE access control stack. In Proxmox VE 8.1, two major milestones were achieved: firstly, DHCP integration was added to the IP address management (IPAM) feature, and secondly, the SDN integration is now installed by default.

Current Status

The current support status for the various layers of our SDN installation is as follows:

- Core SDN, which includes VNet management and its integration with the Proxmox VE stack, is fully supported.
- IPAM, including DHCP management for virtual guests, is in tech preview.
- Complex routing via FRRouting and controller integration are in tech preview.

Installation

SDN Core

Since Proxmox VE 8.1 the core Software-Defined Network (SDN) packages are installed by default.

If you upgrade from an older version, you need to install the `libpve-network-perl` package on every node:

```
apt update
apt install libpve-network-perl
```

> Proxmox VE version 7.0 and above have the `ifupdown2` package installed by default. If you originally installed your system with an older version, you need to explicitly install the `ifupdown2` package.

After installation, you need to ensure that the following line is present at the end of the `/etc/network/interfaces` configuration file on all nodes, so that the SDN configuration gets included and activated.

```
source /etc/network/interfaces.d/*
```

DHCP IPAM

The DHCP integration into the built-in *PVE* IP Address Management stack currently uses `dnsmasq` for giving out DHCP leases. This is currently opt-in.

To use that feature you need to install the `dnsmasq` package on every node:

```
apt update
apt install dnsmasq
# disable default instance
systemctl disable --now dnsmasq
```

FRRouting

The Proxmox VE SDN stack uses the FRRouting project for advanced setups. This is currently opt-in.

To use the SDN routing integration you need to install the `frr-pythontools` package on all nodes:

```
apt update
apt install frr-pythontools
```

Configuration Overview

Configuration is done at the web UI at datacenter level, separated into the following sections:

- SDN:: Here you get an overview of the current active SDN state, and you can apply all pending changes to the whole cluster.
- Zones: Create and manage the virtually separated network zones
- VNets VNets: Create virtual network bridges and manage subnets

The Options category allows adding and managing additional services to be used in your SDN setup.

- Controllers: For controlling layer 3 routing in complex setups
- DHCP: Define a DHCP server for a zone that automatically allocates IPs for guests in the IPAM and leases them to the guests via DHCP.
- IPAM: Enables external for IP address management for guests
- DNS: Define a DNS server integration for registering virtual guests' hostname and IP addresses

Technology & Configuration

The Proxmox VE Software-Defined Network implementation uses standard Linux networking as much as possible. The reason for this is that modern Linux networking provides almost all needs for a feature full SDN implementation and avoids adding external dependencies and reduces the overall amount of components that can break.

The Proxmox VE SDN configurations are located in `/etc/pve/sdn`, which is shared with all other cluster nodes through the Proxmox VE configuration file system. Those configurations get translated to the respective configuration formats of the tools that manage the underlying network stack (for example `ifupdown2` or `frr`).

New changes are not immediately applied but recorded as pending first. You can then apply a set of different changes all at once in the main *SDN* overview panel on the web interface. This system allows to roll-out various changes as single atomic one.

The SDN tracks the rolled-out state through the *.running-config* and *.version* files located in */etc/pve/sdn*.

Zones

A zone defines a virtually separated network. Zones are restricted to specific nodes and assigned permissions, in order to restrict users to a certain zone and its contained VNets.

Different technologies can be used for separation:

- Simple: Isolated Bridge. A simple layer 3 routing bridge (NAT)
- VLAN: Virtual LANs are the classic method of subdividing a LAN
- QinQ: Stacked VLAN (formally known as `IEEE 802.1ad`)
- VXLAN: Layer 2 VXLAN network via a UDP tunnel
- EVPN (BGP EVPN): VXLAN with BGP to establish Layer 3 routing

Common Options

The following options are available for all zone types:

Nodes
: The nodes which the zone and associated VNets should be deployed on.

IPAM
: Use an IP Address Management (IPAM) tool to manage IPs in the zone. Optional, defaults to `pve`.

DNS
: DNS API server. Optional.

ReverseDNS
: Reverse DNS API server. Optional.

DNSZone
: DNS domain name. Used to register hostnames, such as `<hostname>.<domain>`. The DNS zone must already exist on the DNS server. Optional.

Simple Zones

This is the simplest plugin. It will create an isolated VNet bridge. This bridge is not linked to a physical interface, and VM traffic is only local on each the node. It can be used in NAT or routed setups.

VLAN Zones

The VLAN plugin uses an existing local Linux or OVS bridge to connect to the node's physical interface. It uses VLAN tagging defined in the VNet to isolate the network segments. This allows connectivity of VMs between different nodes.

VLAN zone configuration options:

Bridge
: The local bridge or OVS switch, already configured on **each** node that allows node-to-node connection.

QinQ Zones

QinQ also known as VLAN stacking, that uses multiple layers of VLAN tags for isolation. The QinQ zone defines the outer VLAN tag (the *Service VLAN*) whereas the inner VLAN tag is defined by the VNet.

> Your physical network switches must support stacked VLANs for this configuration.

QinQ zone configuration options:

Bridge
: A local, VLAN-aware bridge that is already configured on each local node

Service VLAN
: The main VLAN tag of this zone

Service VLAN Protocol
: Allows you to choose between an 802.1q (default) or 802.1ad service VLAN type.

MTU
: Due to the double stacking of tags, you need 4 more bytes for QinQ VLANs. For example, you must reduce the MTU to `1496` if you physical interface MTU is `1500`.

VXLAN Zones

The VXLAN plugin establishes a tunnel (overlay) on top of an existing network (underlay). This encapsulates layer 2 Ethernet frames within layer 4 UDP datagrams using the default destination port `4789`.

You have to configure the underlay network yourself to enable UDP connectivity between all peers.

You can, for example, create a VXLAN overlay network on top of public internet, appearing to the VMs as if they share the same local Layer 2 network.

> VXLAN on its own does does not provide any encryption. When joining multiple sites via VXLAN, make sure to establish a secure connection between the site, for example by using a site-to-site VPN.

VXLAN zone configuration options:

Peers Address List
: A list of IP addresses of each node in the VXLAN zone. This can be external nodes reachable at this IP address. All nodes in the cluster need to be mentioned here.

MTU
: Because VXLAN encapsulation uses 50 bytes, the MTU needs to be 50 bytes lower than the outgoing physical interface.

EVPN Zones

The EVPN zone creates a routable Layer 3 network, capable of spanning across multiple clusters. This is achieved by establishing a VPN and utilizing BGP as the routing protocol.

The VNet of EVPN can have an anycast IP address and/or MAC address. The bridge IP is the same on each node, meaning a virtual guest can use this address as gateway.

Routing can work across VNets from different zones through a VRF (Virtual Routing and Forwarding) interface.

EVPN zone configuration options:

VRF VXLAN ID
: A VXLAN-ID used for dedicated routing interconnect between VNets. It must be different than the VXLAN-ID of the VNets.

Controller
: The EVPN-controller to use for this zone. (See controller plugins section).

VNet MAC Address
: Anycast MAC address that gets assigned to all VNets in this zone. Will be auto-generated if not defined.

Exit Nodes
: Nodes that shall be configured as exit gateways from the EVPN network, through the real network. The configured nodes will announce a default route in the EVPN network. Optional.

Primary Exit Node
: If you use multiple exit nodes, force traffic through this primary exit node, instead of load-balancing on all nodes. Optional but necessary if you want to use SNAT or if your upstream router doesn’t support ECMP.

Exit Nodes Local Routing
: This is a special option if you need to reach a VM/CT service from an exit node. (By default, the exit nodes only allow forwarding traffic between real network and EVPN network). Optional.

Advertise Subnets
: Announce the full subnet in the EVPN network. If you have silent VMs/CTs (for example, if you have multiple IPs and the anycast gateway doesn’t see traffic from theses IPs, the IP addresses won’t be able to be reached inside the EVPN network). Optional.

Disable ARP ND Suppression
: Don’t suppress ARP or ND (Neighbor Discovery) packets. This is required if you use floating IPs in your VMs (IP and MAC addresses are being moved between systems). Optional.

Route-target Import
: Allows you to import a list of external EVPN route targets. Used for cross-DC or different EVPN network interconnects. Optional.

MTU
: Because VXLAN encapsulation uses 50 bytes, the MTU needs to be 50 bytes less than the maximal MTU of the outgoing physical interface. Optional, defaults to 1450.

VNets

After creating a virtual network (VNet) through the SDN GUI, a local network interface with the same name is available on each node. To connect a guest to the VNet, assign the interface to the guest and set the IP address accordingly.

Depending on the zone, these options have different meanings and are explained in the respective zone section in this document.

> In the current state, some options may have no effect or won’t work in certain zones.

VNet configuration options:

ID
: An up to 8 character ID to identify a VNet

Comment
: More descriptive identifier. Assigned as an alias on the interface. Optional

Zone
: The associated zone for this VNet

Tag
: The unique VLAN or VXLAN ID

VLAN Aware
: Enables vlan-aware option on the interface, enabling configuration in the guest.

Subnets

A subnet define a specific IP range, described by the CIDR network address. Each VNet, can have one or more subnets.

A subnet can be used to:

- Restrict the IP addresses you can define on a specific VNet
- Assign routes/gateways on a VNet in layer 3 zones
- Enable SNAT on a VNet in layer 3 zones
- Auto assign IPs on virtual guests (VM or CT) through IPAM plugins
- DNS registration through DNS plugins

If an IPAM server is associated with the subnet zone, the subnet prefix will be automatically registered in the IPAM.

Subnet configuration options:

ID
: A CIDR network address, for example 10.0.0.0/8

Gateway
: The IP address of the network’s default gateway. On layer 3 zones (Simple/EVPN plugins), it will be deployed on the VNet.

SNAT
: Enable Source NAT which allows VMs from inside a VNet to connect to the outside network by forwarding the packets to the nodes outgoing interface. On EVPN zones, forwarding is done on EVPN gateway-nodes. Optional.

DNS Zone Prefix
: Add a prefix to the domain registration, like <hostname>.prefix.<domain> Optional.

Controllers

Some zones implement a separated control and data plane that require an external controller to manage the VNet’s control plane.

Currently, only the `EVPN` zone requires an external controller.

EVPN Controller

The `EVPN`, zone requires an external controller to manage the control plane. The EVPN controller plugin configures the Free Range Routing (frr) router.

To enable the EVPN controller, you need to install frr on every node that shall participate in the EVPN zone.

```
apt install frr frr-pythontools
```

EVPN controller configuration options:

ASN #
: A unique BGP ASN number. It’s highly recommended to use a private ASN number (64512 – 65534, 4200000000 – 4294967294), as otherwise you could end up breaking global routing by mistake.

Peers
: An IP list of all nodes that are part of the EVPN zone. (could also be external nodes or route reflector servers)

BGP Controller

The BGP controller is not used directly by a zone. You can use it to configure FRR to manage BGP peers.

For BGP-EVPN, it can be used to define a different ASN by node, so doing EBGP. It can also be used to export EVPN routes to an external BGP peer.

By default, for a simple full mesh EVPN, you don’t need to define a BGP controller.

BGP controller configuration options:

Node
: The node of this BGP controller

ASN #
: A unique BGP ASN number. It’s highly recommended to use a private ASN number in the range (64512 - 65534) or (4200000000 - 4294967294), as otherwise you could break global routing by mistake.

Peer
: A list of peer IP addresses you want to communicate with using the underlying BGP network.

EBGP
: If your peer’s remote-AS is different, this enables EBGP.

Loopback Interface
: Use a loopback or dummy interface as the source of the EVPN network (for multipath).

ebgp-mutltihop
: Increase the number of hops to reach peers, in case they are not directly connected or they use loopback.

bgp-multipath-as-path-relax
: Allow ECMP if your peers have different ASN.

ISIS Controller

The ISIS controller is not used directly by a zone. You can use it to configure FRR to export EVPN routes to an ISIS domain.

ISIS controller configuration options:

Node
: The node of this ISIS controller.

Domain
: A unique ISIS domain.

Network Entity Title
: A Unique ISIS network address that identifies this node.

Interfaces
: A list of physical interface(s) used by ISIS.

Loopback
: Use a loopback or dummy interface as the source of the EVPN network (for multipath).

IPAM

IP Address Management (IPAM) tools manage the IP addresses of clients on the network. SDN in Proxmox VE uses IPAM for example to find free IP addresses for new guests.

A single IPAM instance can be associated with one or more zones.

PVE IPAM Plugin

The default built-in IPAM for your Proxmox VE cluster.

You can inspect the current status of the PVE IPAM Plugin via the IPAM panel in the SDN section of the datacenter configuration. This UI can be used to create, update and delete IP mappings. This is particularly convenient in conjunction with the DHCP feature.

If you are using DHCP, you can use the IPAM panel to create or edit leases for specific VMs, which enables you to change the IPs allocated via DHCP. When editing an IP of a VM that is using DHCP you must make sure to force the guest to acquire a new DHCP leases. This can usually be done by reloading the network stack of the guest or rebooting it.

NetBox IPAM Plugin

NetBox is an open-source IP Address Management (IPAM) and datacenter infrastructure management (DCIM) tool.

To integrate NetBox with Proxmox VE SDN, create an API token in NetBox as described here: https://docs.netbox.dev/en/stable/integrations/rest-api/#tokens

The NetBox configuration properties are:

URL
: The NetBox REST API endpoint: `http://yournetbox.domain.com/api`

Token
: An API access token

phpIPAM Plugin

In phpIPAM you need to create an "application" and add an API token with admin privileges to the application.

The phpIPAM configuration properties are:

URL
: The REST-API endpoint: `http://phpipam.domain.com/api/<appname>/`

Token

An API access token

Section
: An integer ID. Sections are a group of subnets in phpIPAM. Default installations use `sectionid=1` for customers.

DNS

The DNS plugin in Proxmox VE SDN is used to define a DNS API server for registration of your hostname and IP address. A DNS configuration is associated with one or more zones, to provide DNS registration for all the subnet IPs configured for a zone.

PowerDNS Plugin

https://doc.powerdns.com/authoritative/http-api/index.html

You need to enable the web server and the API in your PowerDNS config:

```
api=yes
api-key=arandomgeneratedstring
webserver=yes
webserver-port=8081
```

The PowerDNS configuration options are:

url
: The REST API endpoint: http://yourpowerdnserver.domain.com:8081/api/v1/servers/localhost

key
: An API access key

ttl
: The default TTL for records

DHCP

The DHCP plugin in Proxmox VE SDN can be used to automatically deploy a DHCP server for a Zone. It provides DHCP for all Subnets in a Zone that have a DHCP range configured. Currently the only available backend plugin for DHCP is the dnsmasq plugin.

The DHCP plugin works by allocating an IP in the IPAM plugin configured in the Zone when adding a new network interface to a VM/CT. You can find more information on how to configure an IPAM in the respective section of our documentation.

When the VM starts, a mapping for the MAC address and IP gets created in the DHCP plugin of the zone. When the network interfaces is removed or the VM/CT are destroyed, then the entry in the IPAM and the DHCP server are deleted as well.

> Some features (adding/editing/removing IP mappings) are currently only available when using the PVE IPAM plugin.

Configuration

You can enable automatic DHCP for a zone in the Web UI via the Zones panel and enabling DHCP in the advanced options of a zone.

> Currently only Simple Zones have support for automatic DHCP

After automatic DHCP has been enabled for a Zone, DHCP Ranges need to be configured for the subnets in a Zone. In order to that, go to the Vnets panel and select the Subnet for which you want to configure DHCP ranges. In the edit dialogue you can configure DHCP ranges in the respective Tab. Alternatively you can set DHCP ranges for a Subnet via the following CLI command:

```
pvesh set
/cluster/sdn/vnets/<vnet>/subnets/<subnet>
 -dhcp-range start-address=10.0.1.100,end-
address=10.0.1.200
 -dhcp-range start-address=10.0.2.100,end-
address=10.0.2.200
```

You also need to have a gateway configured for the subnet - otherwise automatic DHCP will not work.

The DHCP plugin will then allocate IPs in the IPAM only in the configured ranges.

Do not forget to follow the installation steps for the dnsmasq DHCP plugin as well.

Plugins

Dnsmasq Plugin

Currently this is the only DHCP plugin and therefore the plugin that gets used when you enable DHCP for a zone.

Installation

For installation see the DHCP IPAM section.

Configuration

The plugin will create a new systemd service for each zone that dnsmasq gets deployed to. The name for the service is `dnsmasq@<zone>`. The lifecycle of this service is managed by the DHCP plugin.

The plugin automatically generates the following configuration files in the folder `/etc/dnsmasq.d/<zone>`:

`00-default.conf`
: This contains the default global configuration for a dnsmasq instance.

`10-<zone>-<subnet_cidr>.conf`
: This file configures specific options for a subnet, such as the DNS server that should get configured via DHCP.

`10-<zone>-<subnet_cidr>.ranges.conf`
: This file configures the DHCP ranges for the dnsmasq instance.

`ethers`
: This file contains the MAC-address and IP mappings from the IPAM plugin. In order to override those mappings, please use the respective IPAM plugin rather than editing this file, as it will get overwritten by the dnsmasq plugin.

You must not edit any of the above files, since they are managed by the DHCP plugin. In order to customize the dnsmasq configuration you can create additional files (e.g. `90-custom.conf`) in the configuration folder - they will not get changed by the dnsmasq DHCP plugin.

Configuration files are read in order, so you can control the order of the configuration directives by naming your custom configuration files appropriately.

DHCP leases are stored in the file `/var/lib/misc/dnsmasq.<zone>.leases`.

When using the PVE IPAM plugin, you can update, create and delete DHCP leases. For more information please consult the documentation of the PVE IPAM plugin. Changing DHCP leases is currently not supported for the other IPAM plugins.

Examples

This section presents multiple configuration examples tailored for common SDN use cases. It aims to offer tangible implementations, providing additional details to enhance comprehension of the available configuration options.

Simple Zone Example

Simple zone networks create an isolated network for guests on a single host to connect to each other.

> connection between guests are possible if all guests reside on a same host but cannot be reached on other nodes.

- Create a simple zone named `simple`.
- Add a VNet names `vnet1`.
- Create a Subnet with a gateway and the SNAT option enabled.
- This creates a network bridge `vnet1` on the node. Assign this bridge to the guests that shall join the network and configure an IP address.

The network interface configuration in two VMs may look like this which allows them to communicate via the 10.0.1.0/24 network.

```
allow-hotplug ens19
iface ens19 inet static
        address 10.0.1.14/24
```

```
allow-hotplug ens19
iface ens19 inet static
        address 10.0.1.15/24
```

Source NAT Example

If you want to allow outgoing connections for guests in the simple network zone the simple zone offers a Source NAT (SNAT) option.

Starting from the configuration above, Add a Subnet to the VNet `vnet1`, set a gateway IP and enable the SNAT option.

```
Subnet: 172.16.0.0/24
Gateway: 172.16.0.1
SNAT: checked
```

In the guests configure the static IP address inside the subnet's IP range.

The node itself will join this network with the Gateway IP *172.16.0.1* and function as the NAT gateway for guests within the subnet range.

VLAN Setup Example

When VMs on different nodes need to communicate through an isolated network, the VLAN zone allows network level isolation using VLAN tags.

Create a VLAN zone named `myvlanzone`:

```
ID: myvlanzone
Bridge: vmbr0
```

Create a VNet named `myvnet1` with VLAN tag 10 and the previously created `myvlanzone`.

```
ID: myvnet1
Zone: myvlanzone
Tag: 10
```

Apply the configuration through the main SDN panel, to create VNets locally on each node.

Create a Debian-based virtual machine (*vm1*) on node1, with a vNIC on `myvnet1`.

Use the following network configuration for this VM:

```
auto eth0
iface eth0 inet static
        address 10.0.3.100/24
```

Create a second virtual machine (*vm2*) on node2, with a vNIC on the same VNet `myvnet1` as vm1.

Use the following network configuration for this VM:

```
auto eth0
iface eth0 inet static
        address 10.0.3.101/24
```

Following this, you should be able to ping between both VMs using that network.

QinQ Setup Example

This example configures two QinQ zones and adds two VMs to each zone to demonstrate the additional layer of VLAN tags which allows the configuration of more isolated VLANs.

A typical use case for this configuration is a hosting provider that provides an isolated network to customers for VM communication but isolates the VMs from other customers.

Create a QinQ zone named `qinqzone1` with service VLAN 20

```
ID: qinqzone1
Bridge: vmbr0
Service VLAN: 20
```

Create another QinQ zone named `qinqzone2` with service VLAN 30

```
ID: qinqzone2
Bridge: vmbr0
```

```
Service VLAN: 30
```

Create a VNet named `myvnet1` with VLAN-ID 100 on the previously created `qinqzone1` zone.

```
ID: qinqvnet1
Zone: qinqzone1
Tag: 100
```

Create a `myvnet2` with VLAN-ID 100 on the `qinqzone2` zone.

```
ID: qinqvnet2
Zone: qinqzone2
Tag: 100
```

Apply the configuration on the main SDN web interface panel to create VNets locally on each node.

Create four Debian-bases virtual machines (vm1, vm2, vm3, vm4) and add network interfaces to vm1 and vm2 with bridge `qinqvnet1` and vm3 and vm4 with bridge `qinqvnet2`.

Inside the VM, configure the IP addresses of the interfaces, for example via `/etc/network/interfaces`:

```
auto eth0
iface eth0 inet static
        address 10.0.3.101/24
```

Configure all four VMs to have IP addresses from the *10.0.3.101* to *10.0.3.104* range.

Now you should be able to ping between the VMs *vm1* and *vm2*, as well as between *vm3* and *vm4*. However, neither of VMs *vm1* or *vm2* can ping VMs *vm3* or *vm4*, as they are on a different zone with a different service-VLAN.

VXLAN Setup Example

The example assumes a cluster with three nodes, with the node IP addresses 192.168.0.1, 192.168.0.2 and 192.168.0.3.

Create a VXLAN zone named `myvxlanzone` and add all IPs from the nodes to the peer address list. Use the default MTU of 1450 or configure accordingly.

```
ID: myvxlanzone
Peers Address List:
192.168.0.1,192.168.0.2,192.168.0.3
```

Create a VNet named `vxvnet1` using the VXLAN zone `myvxlanzone` created previously.

```
ID: vxvnet1
Zone: myvxlanzone
Tag: 100000
```

Apply the configuration on the main SDN web interface panel to create VNets locally on each nodes.

Create a Debian-based virtual machine (*vm1*) on node1, with a vNIC on `vxvnet1`.

Use the following network configuration for this VM (note the lower MTU).

```
auto eth0
iface eth0 inet static
        address 10.0.3.100/24
        mtu 1450
```

Create a second virtual machine (*vm2*) on node3, with a vNIC on the same VNet `vxvnet1` as vm1.

Use the following network configuration for this VM:

```
auto eth0
iface eth0 inet static
        address 10.0.3.101/24
        mtu 1450
```

Then, you should be able to ping between between *vm1* and *vm2*.

EVPN Setup Example

The example assumes a cluster with three nodes (node1, node2, node3) with IP addresses 192.168.0.1, 192.168.0.2 and 192.168.0.3.

Create an EVPN controller, using a private ASN number and the above node addresses as peers.

```
ID: myevpnctl
ASN#: 65000
Peers: 192.168.0.1,192.168.0.2,192.168.0.3
```

Create an EVPN zone named `myevpnzone`, assign the previously created EVPN-controller and define *node1* and *node2* as exit nodes.

```
ID: myevpnzone
VRF VXLAN Tag: 10000
Controller: myevpnctl
MTU: 1450
VNet MAC Address: 32:F4:05:FE:6C:0A
Exit Nodes: node1,node2
```

Create the first VNet named `myvnet1` using the EVPN zone `myevpnzone`.

```
ID: myvnet1
Zone: myevpnzone
Tag: 11000
```

Create a subnet on `myvnet1`:

```
Subnet: 10.0.1.0/24
Gateway: 10.0.1.1
```

Create the second VNet named `myvnet2` using the same EVPN zone `myevpnzone`.

```
ID: myvnet2
Zone: myevpnzone
Tag: 12000
```

Create a different subnet on `myvnet2``:

```
Subnet: 10.0.2.0/24
Gateway: 10.0.2.1
```

Apply the configuration from the main SDN web interface panel to create VNets locally on each node and generate the FRR configuration.

Create a Debian-based virtual machine (*vm1*) on node1, with a vNIC on `myvnet1`.

Use the following network configuration for *vm1*:

```
auto eth0
iface eth0 inet static
        address 10.0.1.100/24
        gateway 10.0.1.1
        mtu 1450
```

Create a second virtual machine (*vm2*) on node2, with a vNIC on the other VNet `myvnet2`.

Use the following network configuration for *vm2*:

```
auto eth0
iface eth0 inet static
        address 10.0.2.100/24
        gateway 10.0.2.1
        mtu 1450
```

Now you should be able to ping vm2 from vm1, and vm1 from vm2.

If you ping an external IP from *vm2* on the non-gateway node3, the packet will go to the configured *myvnet2* gateway, then will be routed to the exit nodes (*node1* or *node2*) and from there it will leave those nodes over the default gateway configured on node1 or node2.

You need to add reverse routes for the *10.0.1.0/24* and *10.0.2.0/24* networks to node1 and node2 on your external gateway, so that the public network can reply back.

If you have configured an external BGP router, the BGP-EVPN routes (10.0.1.0/24 and 10.0.2.0/24 in this example), will be announced dynamically.

Notes

Multiple EVPN Exit Nodes

If you have multiple gateway nodes, you should disable the `rp_filter` (Strict Reverse Path Filter) option, because packets can arrive at one node but go out from another node.

Add the following to `/etc/sysctl.conf`:

```
net.ipv4.conf.default.rp_filter=0
net.ipv4.conf.all.rp_filter=0
```

VXLAN IPSEC Encryption

To add IPSEC encryption on top of a VXLAN, this example shows how to use `strongswan`.

You`ll need to reduce the *MTU* by additional 60 bytes for IPv4 or 80 bytes for IPv6 to handle encryption.

So with default real 1500 MTU, you need to use a MTU of 1370 (1370 + 80 (IPSEC) + 50 (VXLAN) == 1500).

Install strongswan on the host.

```
apt install strongswan
```

Add configuration to `/etc/ipsec.conf`. We only need to encrypt traffic from the VXLAN UDP port *4789*.

```
conn %default
    ike=aes256-sha1-modp1024!  # the fastest, but reasonably secure cipher on modern HW
    esp=aes256-sha1!
    leftfirewall=yes           # this is necessary when using Proxmox VE firewall rules

conn output
    rightsubnet=%dynamic[udp/4789]
    right=%any
    type=transport
    authby=psk
    auto=route

conn input
    leftsubnet=%dynamic[udp/4789]
    type=transport
    authby=psk
    auto=route
```

Generate a pre-shared key with:

```
openssl rand -base64 128
```

and add the key to `/etc/ipsec.secrets`, so that the file contents looks like:

```
: PSK <generatedbase64key>
```

Copy the PSK and the configuration to all nodes participating in the VXLAN network.

Version 8.2.2
Last updated Thu Apr 25 09:24:16 CEST 2024