Ansible EVPN/VXLAN Documentation

Release 0.2.0

Damien Garros

Oct 27, 2017

Contents

Sample project using Ansible and Jinja2 template to generate configurations and manage Juniper devices deployed in EVPN/VXLAN Fabric mode.

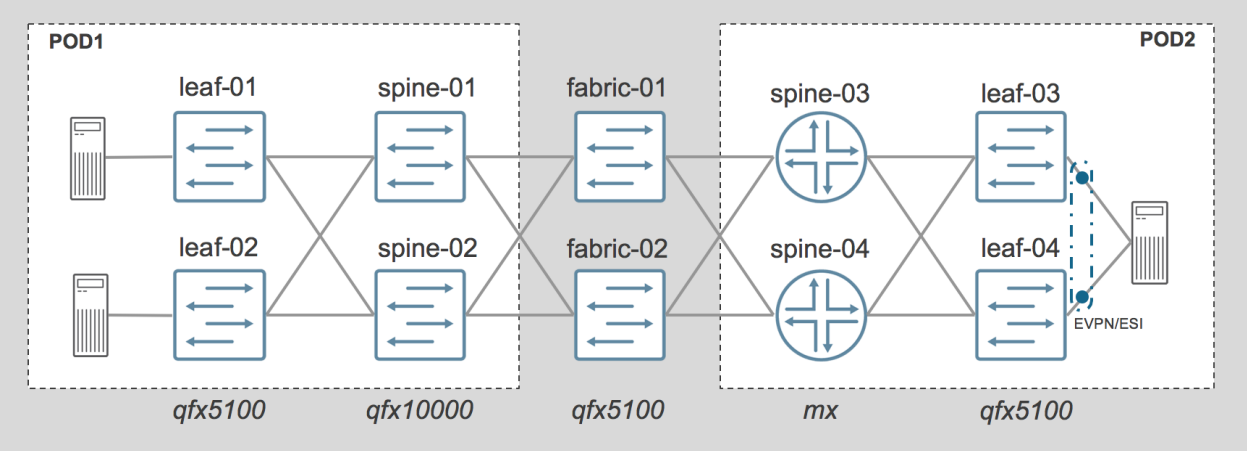
In this project you'll find:

- **Sample project for ansible** with Playbooks and variables to generate EVPN/VXLAN configuration for multi-pod EVPN/Fabric in a multi-tenants environment.
- Examples of configuration EVPN/VXLAN for QFX5k, QFX10k & MX.
- Severals **Jinja2 templates**, packaged and documented into Ansible roles that can be **reuse in other Ansible projects** to easily generate Overlay & Underlay configuration.
- **Playbook to check the health** of an EVPN/VXLAN Fabric.

Contents:

CHAPTER 1

Sample Ansible project to generate EVPN/VXLAN configuration



This project is simulating the creation of a 2 pods EVPN/VXLAN Fabric, POD1 & POD2:

- Each POD is composed of 2 spine and 2 leaf
- PODs are interconnected with 2 qfx5100 acting as Fabric, these are not running EVPN

On POD1

- Spine are QFX10K and leaf are QFX5000
- Leaf are configured with Vlan normalization on their access ports facing servers

On POD2

- Spine are MX480 and leaf are QFX5000
- Leaf are configured with standard trunk interface facing servers,
- 1 server is dual-attached to both leaf using EVPN/ESI and LACP

All devices names, Ip addresses loopback addresses etc .. are defined in the `inventory file named hosts.ini`. All physical connections are defined in the `topology file under group_vars/all`.

Regenerate configurations

Even without real devices, it's possible to regenerate configurations for all devices using ansible playbooks provided with the project

To verify that Ansible & Ansible Junos module for Ansible are properly installed, you can try to regenerate all configs with this command:

```
ansible-playbook pb.conf.all.yaml
```

Note: By default, all configurations generated will be stored under the directory *config/* and will replace existing configuration store there

Scale configurations

The project come with some a solution to easily change the scale of the setup, it's possible to :

- Change the number of tenants
- Change the number of VNI per tenants

To scale the configuration, you need to change some input parameters in the file *group_vars/all/tenant_vni.yaml* *Please refer to instructions in generate-tenant-vni role

Once the input file is modified, you need to regenerate variables first and them regenerate configurations.

```
ansible-playbook pb.generate.variables.yaml
ansible-playbook pb.conf.all.yaml
```

CHAPTER 2

Templates and Roles used to generate configuration

All configurations are generated using Jinja2 templates and variables. To simplificy the management of these templates and make them reusable in other projects, these templates have been pacakges into several roles, each one is generating a part of the final configuration.

All roles are located under the directory roles and are organized as follow

```
underlay-ebgp            # Name of the role
 - README.md            # Documentation and Instructions to reuse
 - meta
 |   - main.yaml        # Indicate author of the project and dependancies
 - defaults
 |   - main.yaml        # Default variables, can be overwritten for each device
 - tasks
 |   - main.yaml        # Action to execute when calling this Roles
 - templates
     - main.conf.j2     # Jinja2 Templates, in most cases, used to generate␣
↪configuration
```

Below the list of roles available, classified per function, with a short description and a link to their respective documentation.

Roles to create the underlay configuration

There are 3 different roles to create an underlay network, only one is needed and all devices must have the same.

Role	Description
underlay-ebgp	Create an Underlay with eBGP with p2p /31 network and 1 ASN per device
underlay-ospf	Create an Underlay with OSPF with p2p /31 network and 1 Area
underlay-ospf-unnumbered	Create an Underlay with OSPF with p2p unnumbered interface and 1 Area

Roles to create the overlay configuration (EVPN)

These roles are complementary and are designed to work together. Each one is specific to a role in the architecture and is specific to device capabilities:

Role	Description
overlay-evpn-qfx-l3	Create iBGP & EVPN configuration for QFX devices that support both L2 & L3 VTEP (QFX10000 today)
overlay-evpn-qfx-l2	Create iBGP & EVPN configuration for QFX devices that only support L2 VTEP (QFX5100/QFX5200)
overlay-evpn-mx-l3	Create iBGP & EVPN configuration for MX devices that only support L2 & L3 VTEP (MX)
overlay-evpn-access	Create access ports configuration to maps existing resources into the overlay (Trunk/LAG/ESI/Vlan mapping)

Roles to generate variables (IPs, vlan)

Role	Description
generate-tenant-vni	Generate variables files to scale Tenant and VNI
generate-p2p-ips	Generate network and ip addresses for P2P links
generate-underlay-bgp	Generate ebgp underlay input variables

Other Roles

Role	Description
common	Generate base configuration
build-config	Assemble all configuration snippet from other roles

CHAPTER 3

Playbooks

All playbooks are stored at the root of the project and are named *pb.*.yaml*

Configure

```
pb.save.config.yaml                 # Download configuration for all devices and save␣
↪them locally

pb.conf.all.yaml                    # Generate and assemble configuration for all␣
↪devices
pb.conf.all.commit.yaml             # Generate, assemble, push and commit configuration␣
↪to all devices

# This project has been updated to use the new Junos modules available in Ansible 2.1
# Some playbooks are also provided with the Junipe.junos modules available in Ansible␣
↪Galaxy.
pb.conf.all.commit.galaxy.yaml      # Generate, assemble, push and commit configuration␣
↪to all devices
                                    # using the Junos modules provided in Ansible Galaxy
```

Test

```
pb.check.connectivity.yaml          # Check if all devices are reachable via Netconf
pb.check.underlay.yaml              # Check the heath of the underlay
pb.check.overlay.yaml               # Check the health of the overlay
```

Generate Variables

```
pb.generate.variables.yaml          # Regenerate variables files for p2p links, Tenants␣
↪and VNI
```

Misc

```
pb.init.make_clean.yaml             # Create temp directory for all devices
```

CHAPTER 4

How to create an Ansible project abstracted from the physical devices

Automation projects for networks devices are often very specific to a particular set of physical devices because of the physical topologies of these devices and how they are connected to each other. There are multiple situations where it's important and very useful to be able to reuse the same automation projects across multiple "network" or group of devices, to name a few:

- Production network with multiple sites
- Configuration validation between production and pre-production networks
- Dynamic/multiple lab environments

Even if you are able to share configuration templates across different projects, there are many information that are difficult to share or reuse. When you want to reuse one project between multiple topology you have to change many information: devices name, interfaces name, IPs etc ...

With Ansible, this challenge is solved using dynamic inventory to dynamically load inventory information and device specific variables. Dynamic inventory is working great for deployment in production but require a back-end system that will store all information. For smaller deployment and for lab most people usually keep their inventory and variables in files stored on the local file system. (TODO, Add link to dynamic inventory) Using static inventory and variables defined in local yaml files, it's possible to make a project easy to reuse across multiple physical topologies.

By using the 3 following steps you'll be able to create to abstract your project from your physical topology:

- Use alias for device in the inventory file
- Centralize all physical information related to a given physical topology in a central topology file
- Define remaining variables specific to a given topology in the inventory file

1/ Use alias for device in the inventory file

The inventory file is one of the main components of Ansible, it contains a lot of information including devices names and groups. The name you use to define your device is very important because it is used in many other places inside ansible. For example, variables are classified per devices using the name defined in the inventory, also available with the variable "inventory_hostname".

Inventory file

```
[spine]
dc1-rack02-qfx10k-01
dc1-rack04-qfx10k-02

[leaf]
dc1-rack08-qfx5100-48s-02
dc1-rack14-qfx5100-48s-05
```

Directory host_vars

```
└─ host_vars
   ├─ dc1-rack02-qfx10k-01.yaml
   ├─ dc1-rack04-qfx10k-02.yaml
   ├─ dc1-rack08-qfx5100-48s-02.yaml
   └─ dc1-rack14-qfx5100-48s-05.yaml
```

It's seam natural to use the real name of your devices in the inventory file but by doing so you'll make it difficult to reuse your project across multiple devices. It's possible to keep your project portable by using an alias in the inventory and create a new variable that contain the real name or IP of the device

Inventory file

```
[spine]
SPINE1 junos_ssh=dc1-rack02-qfx10k-01
SPINE2 junos_ssh=dc1-rack04-qfx10k-02

[leaf]
LEAF02 junos_ssh=dc1-rack08-qfx5100-48s-02
LEAF05 junos_ssh=dc1-rack14-qfx5100-48s-05
```

Directory host_vars

```
└─ host_vars
   ├─ SPINE01.yaml
   ├─ SPINE02.yaml
   ├─ LEAF02.yaml
   └─ LEAF05.yaml
```

2/ Centralize all physical information related to a given physical topology in a central topology file

To prevent very specific information like interfaces name to be duplicated and dispersed everywhere in the project. It possible to centralize them and give them an alias as well.

In this example, I put all information related to my physical layer (interface name, peer etcc ..) in a central file. (sample-topology.yaml) Each interfaces get assigned an alias here: port1, port2 port3. It could be something more meaning full like: to_spine1, to_spine2 All information are stored under a variable named `topo` and each device has it's own section identified with the device name used in the inventory.

Everywhere else, when you need to access the name of an interface you can access it by its variable `{{ topo[inventory_hostname].port1.name }}`

Note: Inventory_hostname is a variable itself and will be automatically replaced with the name of the device used in the inventory, for example: fabric-01

---

hosts.ini

```
[all:vars]
ansible_ssh_user=root
ansible_ssh_pass=<device password>
mgmt_sub_mask=<subnet mask>
netconf_port=830
topology_file=sample-topology.yaml
```

Playbooks

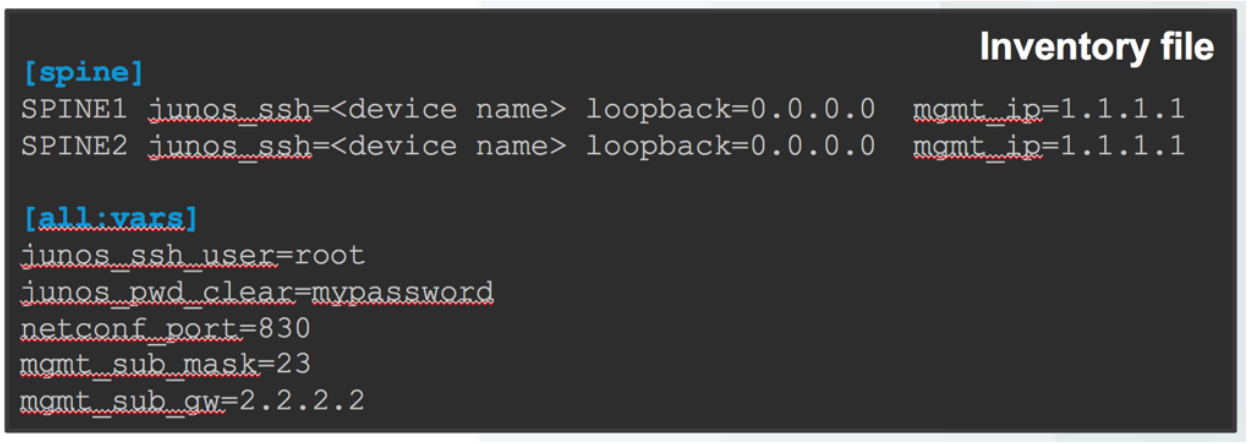
```
- include: pb.conf.all.yaml
- name: Apply configuration
  hosts: all
  connection: local
  gather_facts: no
  pre_tasks:
    - include_vars: "{{ topology_file }}"
  vars:
```

The topology file itself is defined in the inventory file and is loaded at the beginning at each playbook.

Note: The creation of a variable in the inventory file allow to keep the playbook independent of the topology as well

3/ Define remaining variables specific to a given topology in the inventory file

Remaining information that are device specific like management IP, loopback address etc .. can be defined in the inventory file directory. It's possible to define any type of variable in the inventory file, either per device or per group



CHAPTER 5

How to use this project on my own topology

This project has been designed to be easily deploy on multiple topologies, physical or virtual. As much as possible, all information related to a given topology (interface names, device names etc ..) are centralized in 2 files:

The topology file [sample-topology.yaml], this file contains:

- Information required to construct the base configuration (login, dns, ntp etc ..)
- All physical interface names
- Directories to use to generate the configuration

The inventory file [hosts.ini], this file contains:

- Device names
- Device roles in the architecture (using ansible groups)
- Management IP addresses and loopback
- Login, password, management gateway etc ..
- The name of the topology file

When you call an Ansible playbook, you can specify explicitly the inventory file by using the option *-i*

Note: To align, the name of the topology file needs to be define inside the inventory file

```
# Generate configurations for the sample-topology
ansible-playbook -i hosts.ini pb.conf.all.yaml

# Generate configurations for your own topology
ansible-playbook -i mytopology.ini pb.conf.all.yaml
```

It's easy to create your own inventory and topology files to adapt device IP, type and interface names to your environment assuming you have the same base design.

1/ Create your inventory file

The inventory file contains a lot of information and variables but most importantely it define the parsonnality of each device depending on which groups a device belong to.

Different playbooks will be executed for each groups, and each playbook will generate a different part of the configuration.

All devices in the group `spine-mx` will get their configuration from these group - common - underlay-ebgp - overlay-evpn-mx-l3 - build-config

All devices in the group `leaf-qfx-l3` will get their configuration from these group - common - underlay-ebgp - overlay-evpn-qfx-l3 - overlay-evpn-access - build-config

The complete list of role per group is available in the playbook `pb.conf.all.yaml`

Unique ID Each device in the inventory file needs to have a unique ID define inside the variable `id`. This ID is used to automatically generate: - Loopback address - ASN number

2/ Create your own topology file

To properly generate the configuration, it's important to define all information related to your topology in this file: Interface names, dns, login, static route etc ...

Please refer to the documentation of role `generated-underlay-ebgp` to understand how to define in the topology file the information that will be used to generate the underlay

---

Note: if you define `vqfx: true` in the inventory file, DHCP will be automatically configured on the management interface.

3/ Define your IP address plan

You can define your own IP address plan and automatically regenerate all variables by using the playbook `pb.generate.variables.yaml`.

All information can be defined inside the playbook itself in the `vars:` section.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search