In-house automation for ISP and DC networking

Vasilis Stavropoulos
IP Engineering Dpt.
SPARKLE

GRNOG 14

Agenda

- Automating services for local ISP (Sparkle GR - AS198477) and Data Center (DC) networking
- Beginning - early days
- Available Tools
- Evolution
- Integration with a provisioning/orchestration DB (SoT – Netbox)
- Demos

Beginning

- How to start….
- Automation solutions promising many things out of the box
- Reality checks
- Brownfield infrastructures, perhaps with limited support for interaction with automation frameworks
- Vendor solutions vs open-source (Ansible, Python libraries)

Ansible

- Ansible framework was the go-to solution, especially for network engineers
- Ansible existing configuration modules for most major vendors, no demand for scripting knowledge
- Relevant configuration data exist in a yaml file and are consumed by the corresponding ansible module (yaml representation again)
- Debugging difficulties, often errors difficult to interpret, but also lack of experience from our side
- Not very flexible, yaml limitations, partially locked by the predefined modules
- However, easier to start and proceed, especially with templated, stable configurations with not many exceptions (EVPN/VXLAN in our case)

Python framework(s)

- Our SP environment is Juniper based and vendor had already developed Junos PyEZ (Python microframework for managing Junos devices)
- Initial experimental try outs in our SP domain by obtaining facts from devices using ansible and python
- PyEZ library used to execute remote procedure calls (RPC), by establishing a NETCONF session over SSH
- Retrieve configuration data (getter scripts)
- Upload and commit configuration changes (configuration scripts)
- Structured values (by default xml format)

Tables and Views

```
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/19.4R0/junos">
    <interface-information xmlns="http://xml.juniper.net/junos/19.4R0/junos-interface" junos:style="normal">
        <logical-interface>
            <name>ge-1/0/1.0</name>
            <local-index>348</local-index>
            <snmp-index>665</snmp-index>
            <description>CUSTOMER1-MN-0002-IPSTA</description>
            <if-config-flags>
                <iff-device-down/>
                <iff-snmp-traps/>
                <internal-flags>0x0</internal-flags>
            </if-config-flags>
            <encapsulation>ENET2</encapsulation>
            <policer-overhead>
            </policer-overhead>
            <traffic-statistics junos:style="brief">
                <input-packets>0</input-packets>
                <output-packets>0</output-packets>
            </traffic-statistics>
            <filter-information>
            </filter-information>
            <address-family>
                <address-family-name>inet</address-family-name>
                <mtu>1500</mtu>
                <max-local-cache>75000</max-local-cache>
                <new-hold-limit>75000</new-hold-limit>
                <intf-curr-cnt>0</intf-curr-cnt>
                <intf-unresolved-cnt>0</intf-unresolved-cnt>
                <intf-dropcnt>0</intf-dropcnt>
                <address-family-flags>
                    <ifff-sendbcast-pkt-to-re/>
                    <internal-flags>0x2</internal-flags>
                </address-family-flags>
                <interface-address>
                    <ifa-flags>
                        <ifaf-down/>
                        <ifaf-current-preferred/>
                        <ifaf-current-primary/>
                    </ifa-flags>
                    <ifa-destination>37.99.195.76/31</ifa-destination>
                    <ifa-local>37.99.195.76</ifa-local>
                </interface-address>
            </address-family>
            <address-family>
                <address-family-name>multiservice</address-family-name>
                <mtu>Unlimited</mtu>
                <address-family-flags>
                    <internal-flags>0x2</internal-flags>
                </address-family-flags>
            </address-family>
        </logical-interface>
    </interface-information>
```

- PyEZ Tables and Views (yaml representation of command outputs)
- Extract data from specific configuration segments and map them to Python data structures (dictionaries, lists)

```
('ge-1/0/1',
 [('name', 'ge-1/0/1'),
  ('oper', 'down'),
  ('admin', 'up'),
  ('description', 'No Description'),
  ('mtu', 1514),
  ('link_mode', None),
  ('speed', '1000mbps'),
  ('macaddr', 'f8:c0:01:1c:47:09'),
  ('flapped', '2023-01-23 10:32:44 GMT (10w1d 03:23 ago)'),
  ('logical_interface', 'ge-1/0/1.0'),
  ('logical_interface_dsc', 'CUSTOMER1-MN-0002-IPSTA')]),
```

```
 [('intf', 'ge-1/0/1'),
  ('unit_name', '0'),
  ('desc', None),
  ('desc_unit', 'CUSTOMER1-MN-0002-IPSTA'),
  ('address4', '37.99.195.76/31'),
  ('address6', None)]),
```

Initial projects – Skills development

- Fully automate the prefix list update on routers (bgpq3/bash/python)
- Custom on-box python “getters”, combining output of more than one command
- Automating (parts) the most configured services in our network (BGP, static, l2vpn)
- Initial use of NIPAP as IPAM (pynipap library for interfacing from Python) – automatic allocation of subnets
- But that was only for the SP environment
- Another tool for DC IPAM, poor api support, legacy environment

SoT - Netbox

- New DC infrastructure (EVPN/VXLAN) pushed for a more complete solution/approach
- Source of truth (SoT) need for hierarchical representation of regions, sites, devices, interfaces, IPs, vrfs, vlans
- API support for interaction of SoT (netbox) with python scripts
- Insert provisioning data in netbox in a controlled way and generate the configuration
- Focusing on least interaction possible by user during execution to avoid errors and enforce consistency
- Netbox should represent the intended state of each device, regarding the services
- Avoid using many different/scattered scripts
- Gitlab for script (and services) version control

Descriptions - Services - Validation

- Proper description is the core for consistent data
- Simple but meaningful
- Validate and enforce proper interface description in netbox (work in progress…)

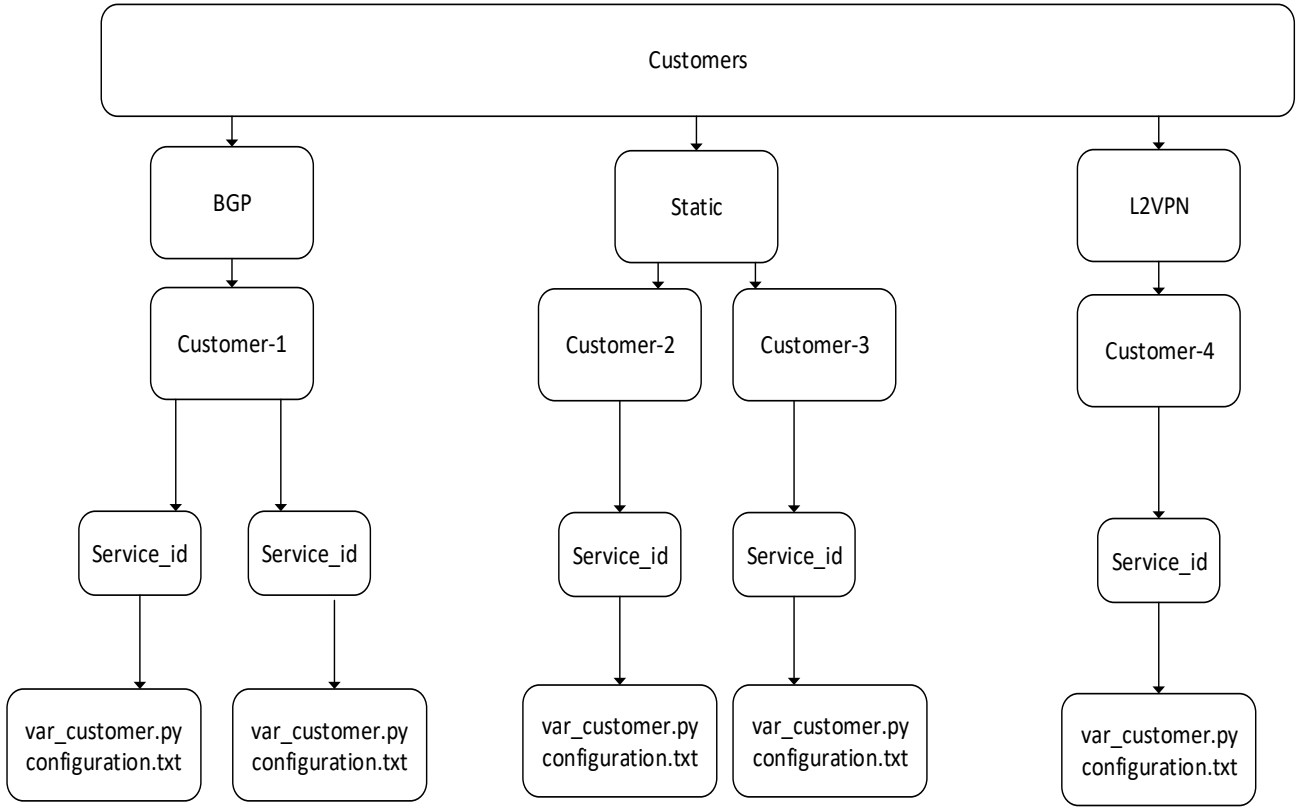
- Description = {Customer}-{service-id}-{service-type}
- {service-type} = IPBGP, IPSTA, L2C……..

- Description contained in every possible segment of the service (interface/protocol/access-list)

Process

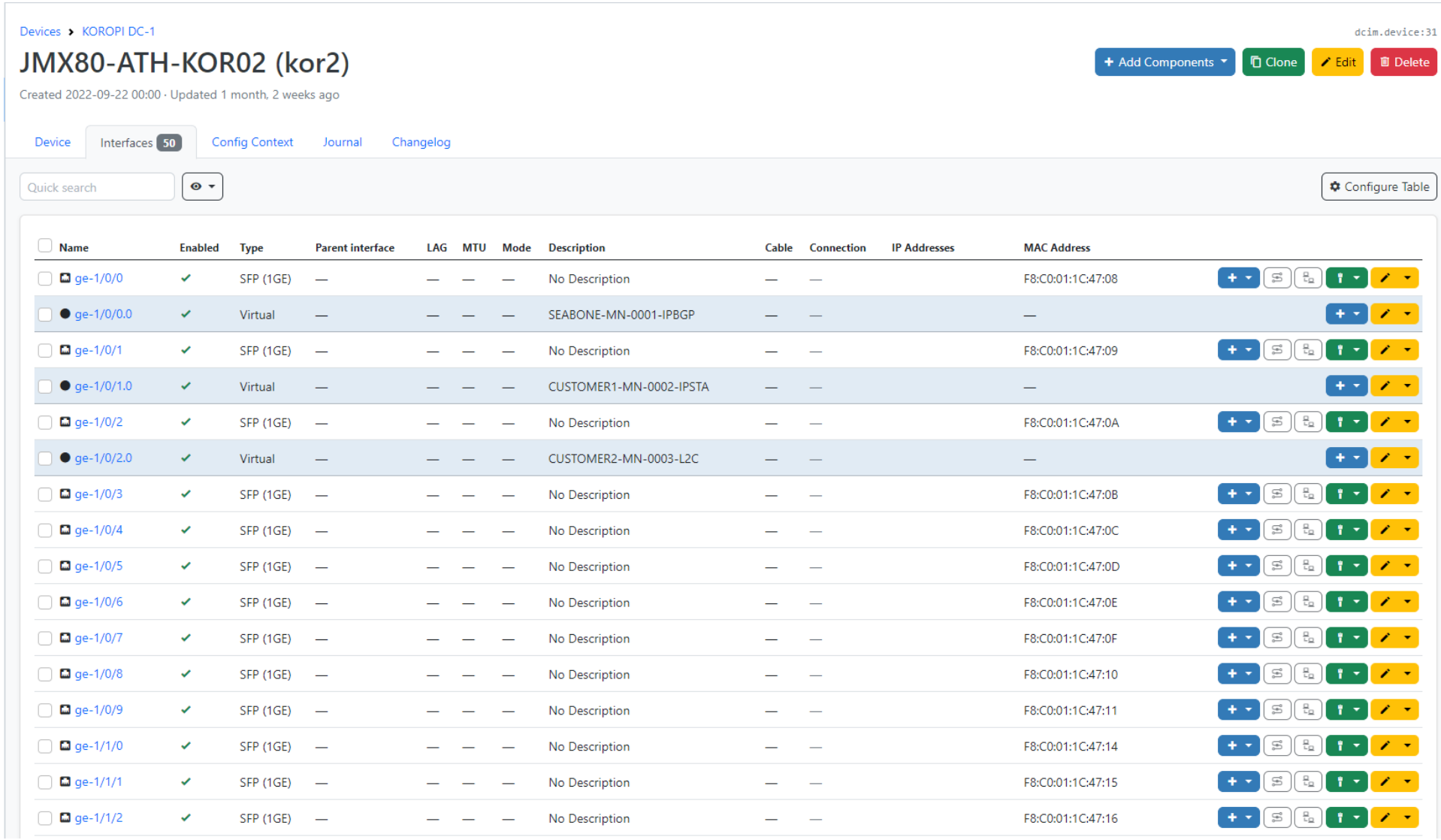
- Services characteristics (p2p or PA subnets, interfaces, asn, as-set, l2 identifiers, etc.) are provisioned through Netbox
- Python script scans Netbox device (or region/site/network) for new services or for services under cancellation
- Programmatic allocation of next available resource (e.g. subnet or vlan) from corresponding pool, via its python API library (pynetbox)
- Configured and generated data from Netbox are consumed by corresponding Jinja templates
- Generated configuration is pushed to the routers
- VM instance “glues" everything together
- Stores scripts/structure/data and communicates as necessary with other applications (e.g., bgpq3)
- Runs post-validation scripts to check consistency (against device or network – Nornir framework)

Service structure



- Hierarchical structure of customers/services/data
- Private GitLab server uploading the structure
- Merge-pull requests could be integrated

Netbox devices and interfaces



Devices › KOROPI DC-1

dcim.device:31

JMX80-ATH-KOR02 (kor2)

+ Add Components | Clone | Edit | Delete

Created 2022-09-22 00:00 · Updated 1 month, 2 weeks ago

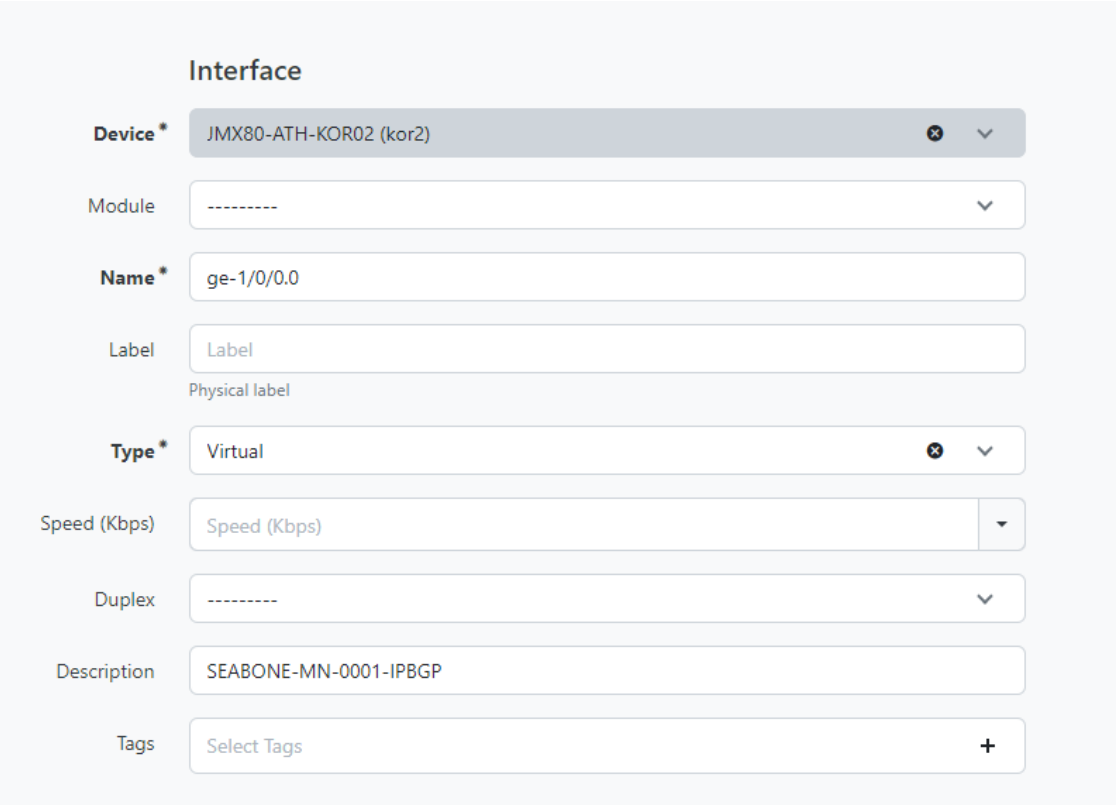
Device | Interfaces 50 | Config Context | Journal | Changelog

Quick search

Configure Table

Name	Enabled	Type	Parent interface	LAG	MTU	Mode	Description	Cable	Connection	IP Addresses	MAC Address
ge-1/0/0	✓	SFP (1GE)	—	—	—	—	No Description	—	—		F8:C0:01:1C:47:08
ge-1/0/0.0	✓	Virtual	—	—	—	—	SEABONE-MN-0001-IPBGP	—	—		—
ge-1/0/1	✓	SFP (1GE)	—	—	—	—	No Description	—	—		F8:C0:01:1C:47:09
ge-1/0/1.0	✓	Virtual	—	—	—	—	CUSTOMER1-MN-0002-IPSTA	—	—		—
ge-1/0/2	✓	SFP (1GE)	—	—	—	—	No Description	—	—		F8:C0:01:1C:47:0A
ge-1/0/2.0	✓	Virtual	—	—	—	—	CUSTOMER2-MN-0003-L2C	—	—		—
ge-1/0/3	✓	SFP (1GE)	—	—	—	—	No Description	—	—		F8:C0:01:1C:47:0B
ge-1/0/4	✓	SFP (1GE)	—	—	—	—	No Description	—	—		F8:C0:01:1C:47:0C
ge-1/0/5	✓	SFP (1GE)	—	—	—	—	No Description	—	—		F8:C0:01:1C:47:0D
ge-1/0/6	✓	SFP (1GE)	—	—	—	—	No Description	—	—		F8:C0:01:1C:47:0E
ge-1/0/7	✓	SFP (1GE)	—	—	—	—	No Description	—	—		F8:C0:01:1C:47:0F
ge-1/0/8	✓	SFP (1GE)	—	—	—	—	No Description	—	—		F8:C0:01:1C:47:10
ge-1/0/9	✓	SFP (1GE)	—	—	—	—	No Description	—	—		F8:C0:01:1C:47:11
ge-1/1/0	✓	SFP (1GE)	—	—	—	—	No Description	—	—		F8:C0:01:1C:47:14
ge-1/1/1	✓	SFP (1GE)	—	—	—	—	No Description	—	—		F8:C0:01:1C:47:15
ge-1/1/2	✓	SFP (1GE)	—	—	—	—	No Description	—	—		F8:C0:01:1C:47:16

Netbox devices and interfaces



- Result of an api call to Netbox for the specific interface
- Python dictionary structure


```
{'_occupied': False,
 'bridge': None,
 'cable': None,
 'cable_end': '',
 'connected_endpoints': None,
 'connected_endpoints_reachable': None,
 'connected_endpoints_type': None,
 'count_fhrp_groups': 0,
 'count_ipaddresses': 0,
 'created': '2023-04-04T12:34:03.055897Z',
 'custom_fields': {'PA': 28, 'PA Allocated': None, 'p2p': 31},
 'description': 'CUSTOMER1-MN-0002-IPSTA',
 'device': {'display': 'JMX80-ATH-KOR02 (kor2)',
            'id': 31,
            'name': 'JMX80-ATH-KOR02',
            'url': 'http://172.16.0.13/api/dcim/devices/31/'},
 'display': 'ge-1/0/1.0',
 'duplex': None,
 'enabled': True,
 'id': 2570,
 'l2vpn_termination': None,
 'label': '',
 'lag': None,
 'last_updated': '2023-04-04T13:16:43.034577Z',
 'link_peers': [],
 'link_peers_type': None,
 'mac_address': None,
 'mark_connected': False,
 'mgmt_only': False,
 'mode': None,
 'module': None,
 'mtu': None,
 'name': 'ge-1/0/1.0',
 'parent': {'_occupied': False,
            'cable': None,
            'device': {'display': 'JMX80-ATH-KOR02 (kor2)',
                       'id': 31,
                       'name': 'JMX80-ATH-KOR02',
                       'url': 'http://172.16.0.13/api/dcim/devices/31/'},
            'display': 'ge-1/0/1',
            'id': 2560,
            'name': 'ge-1/0/1',
            'url': 'http://172.16.0.13/api/dcim/interfaces/2560/'},
 'poe_mode': None,
 'poe_type': None,
 'rf_channel': None,
 'rf_channel_frequency': None,
 'rf_channel_width': None,
 'rf_role': None,
 'speed': None,
 'tagged_vlans': [],
```

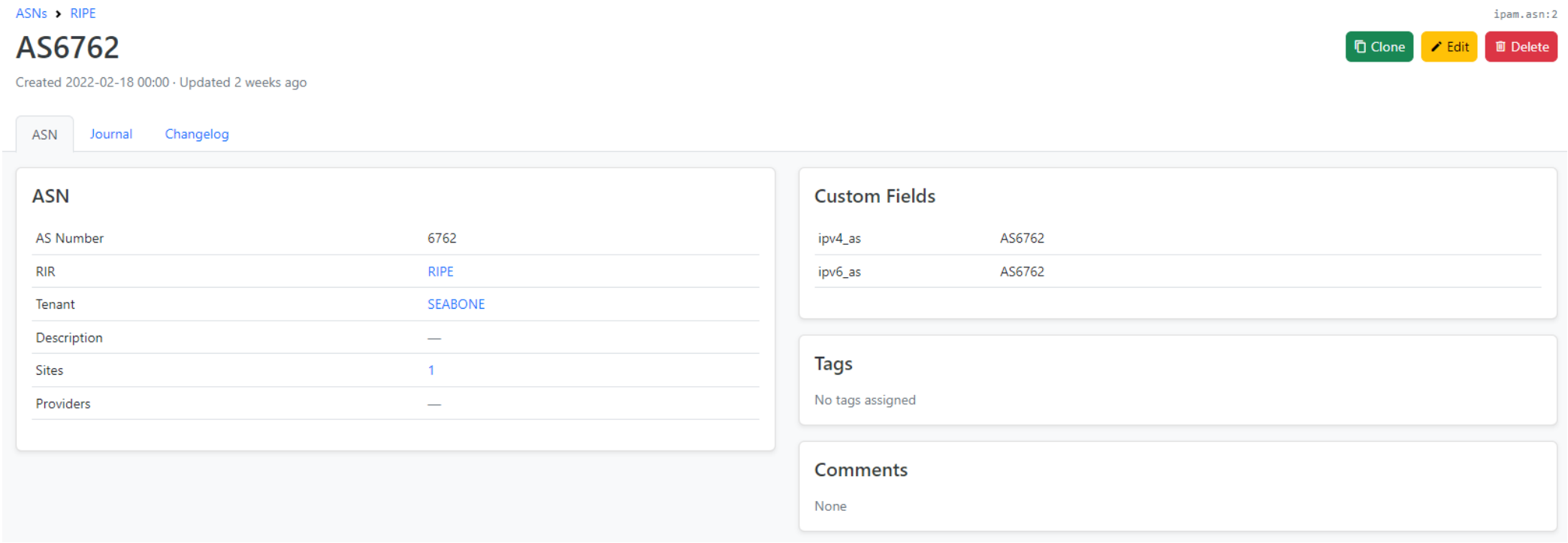

- Normalized result of xml output from Junos device (predefined and custom) PyEz Tables/Views
- Python dictionary structure


```
('ge-1/0/1',
 [('name', 'ge-1/0/1'),
  ('oper', 'down'),
  ('admin', 'up'),
  ('description', 'No Description'),
  ('mtu', 1514),
  ('link_mode', None),
  ('speed', '1000mbps'),
  ('macaddr', 'f8:c0:01:1c:47:09'),
  ('flapped', '2023-01-23 10:32:44 GMT (10w1d 03:23 ago)'),
  ('logical_interface', 'ge-1/0/1.0'),
  ('logical_interface_dsc', 'CUSTOMER1-MN-0002-IPSTA')]),
```

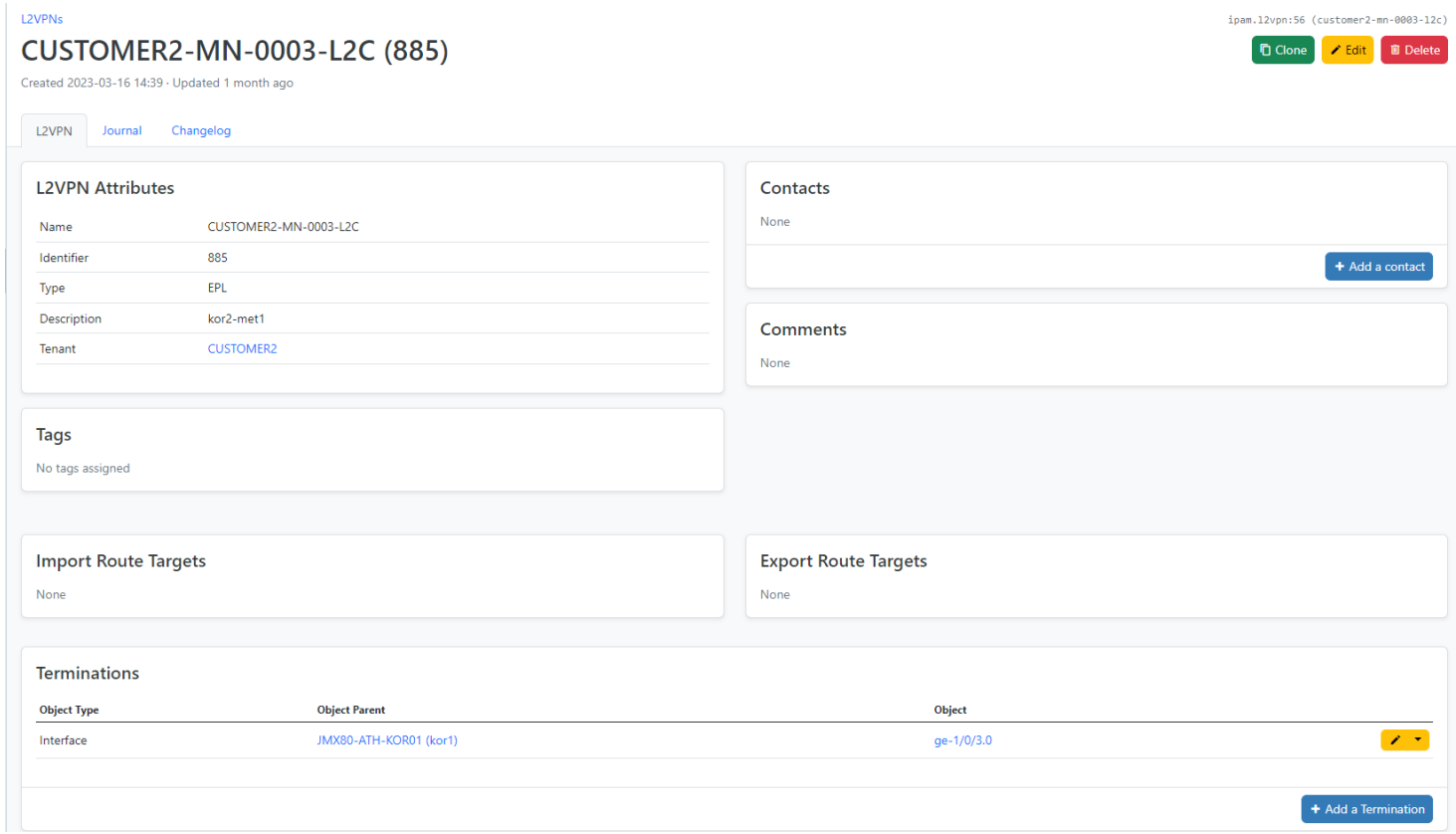


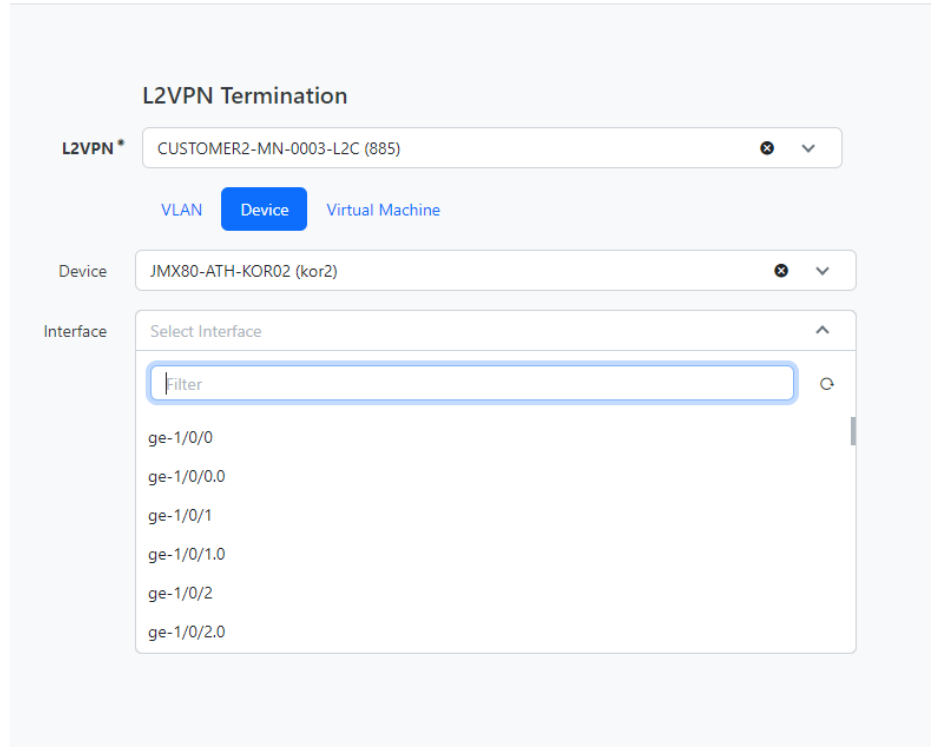
```
 [('intf', 'ge-1/0/1'),
  ('unit_name', '0'),
  ('desc', None),
  ('desc_unit', 'CUSTOMER1-MN-0002-IPSTA'),
  ('address4', '37.99.195.76/31'),
  ('address6', None)]),
```

Netbox services - BGP



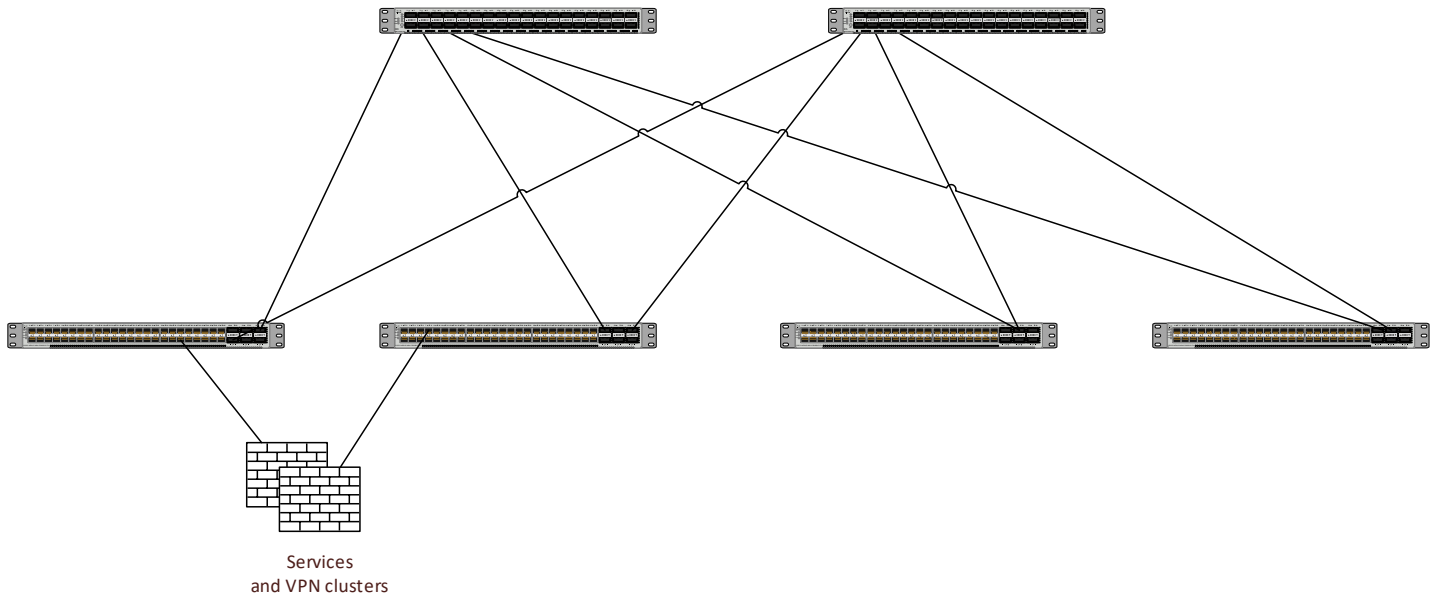
Netbox services – L2VPN





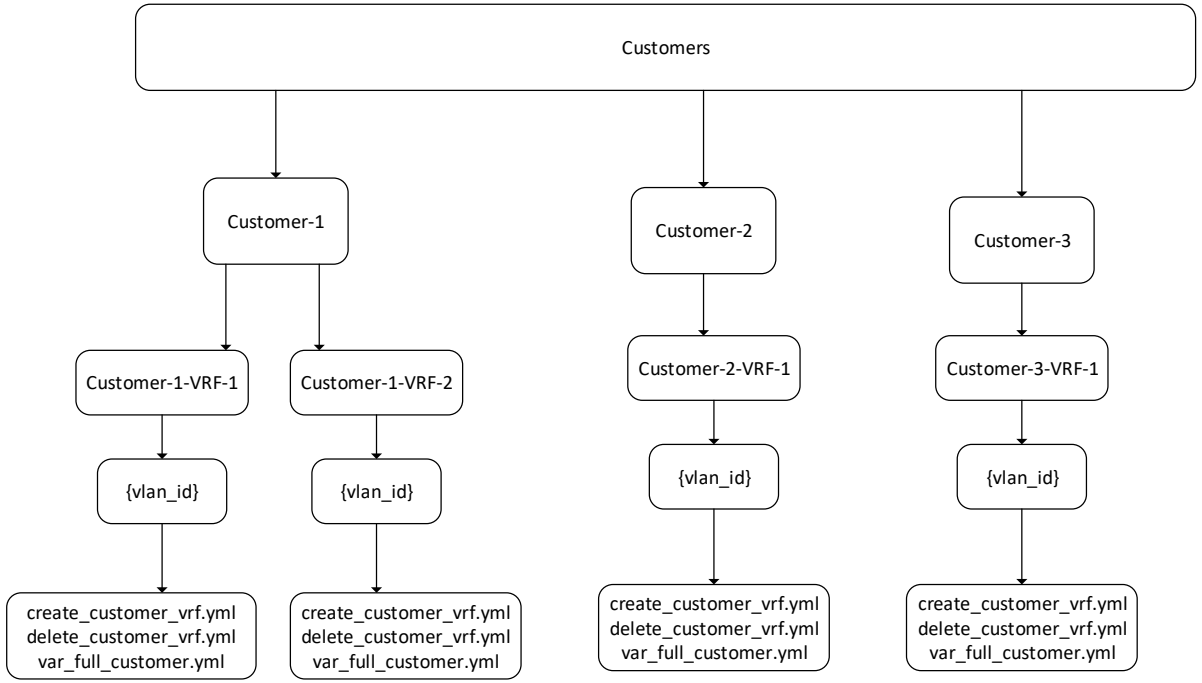
Demo (ISP – Netbox)

DC infrastructure (EVPN/VXLAN)



- Leaf/spine topology (EVPN/VXLAN)
- NX-OS platform
- One common AS number
- IGP (OSFP)
- Per customer variables
  - Service Vlan/vxlan
  - Customer subnet
  - VRF
  - Vlan/vxlan for service and vpn firewalls
  - P2P subnets (FW, VPN)
  - L3vni
- Netbox predefined pools for vlans, subnets per site
- Ansible modules and inventory management for configuration

EVPN/VXLAN configuration



Slightly different approach, two step process :

- Yaml configuration file generated per service by python script
- Ansible playbook for service provisioning on the switches

EVPN/VXLAN configuration

Configuration data

```yaml
---

#Vlan to l3vni
 vlans3:
  - { vlans: 3005, descriptions: CUSTOMER1-VRF-3005-l3vni }
#Vlan to l2vni
 vlans2:
  - { vlans: 123, descriptions: CUSTOMER1-VRF-123-l2vni }
#Vlan to l2vni_firewallGW
 vlansfw:
  - { vlans: 2222, descriptions: CUSTOMER1-VRF-2222-FW }
#Vlan to l2vni_VPNGW
 vlansvpn:
  - { vlans: 2223, descriptions: CUSTOMER1-VRF-2223-VPN }

#L3 interface vlan (MACVRF)
 ipv4s: 10.20.0.129/28
 ipv4sfw: 172.20.0.201/29
 ipv4svpn: 172.20.0.209/29

#L3 vrf
 vrfs: CUSTOMER1-VRF

#Default route configuration
 default_route: 0.0.0.0/0
 next_hop: 172.20.0.202
 cust_net: 10.20.0.128/28
 cust_P2P_FW: 172.20.0.200/29
 cust_P2P_VPN: 172.20.0.208/29
```

Playbook with nxos configuration modules (partial)

```yaml
---
- name: Create L2VNI/L3VNI
  hosts: leaves
  connection: local
  gather_facts: no
  vars_files:
   - var_full_customer.yml

  tasks:
   - name: Create Vlans and map l3vnis
     nxos_vlan:
      vlan_id: "{{ item.vlans }}"
      name: "{{ item.descriptions }}"
      mapped_vni: "60{{ item.vlans }}"
      admin_state: up
     with_items: "{{ vlans3 }}"

   - name: Create Vlans and map l2vnis
     nxos_vlan:
      vlan_id: "{{ item.vlans }}"
      name: "{{ item.descriptions }}"
      mapped_vni: "50{{ item.vlans }}"
      admin_state: up
     with_items: "{{ vlans2 }}"

   - name: Create vrf
     nxos_vrf:
      name: "{{ vrfs }}"
      rd: auto
      vni: "60{{ item.vlans }}"
     with_items: "{{ vlans3 }}"
```

EVPN/VXLAN demo

Thank you !

Q&A