Tutorial: pyGNMI and ChatGPT to troubleshoot EVPN Datacenter Fabrics

JUN-2023

Mau Rojas

bio.site/pinrojas

Abstract


What if I told you
pyGNMI can help you to troubleshoot your datacenter many times faster
And you can try it anywhere with low coding skills

What would I get here?

In this tutorial you will see:

- How to create a network lab using containerlab
- Spine/Leaf + Border-Leaf Architecture
  - Layer 2/3 EVPN-VXLAN
  - eBGP for Underlay
- How to configure multiple devices with GNMIc and Go Templates
- How to script in pyGNMI to troubleshoot EVPN-VXLAN Design

Disclaimer

This presentation is intended for educational purposes only and is not a professional advice. Statements and opinions expressed are those of the presenter and by no means represents the official position nor policies nor formalized way of working with the employer of the presenter.

Presenter's employer does not endorse or approve, and assumes no responsibility for the content, accuracy or completeness of the information presented.

Nothing in this presentation constitutes legal advice, and prior to eventual use of the presented material, the user should obtain appropriate advice in order to be compliant with applicable policies and procedures in user's organization and/or with applicable laws, rules and regulation.

ChatGPT was here (more later)

python function to order array like {['john',30]['smith',34]} descending order using second item in the row.

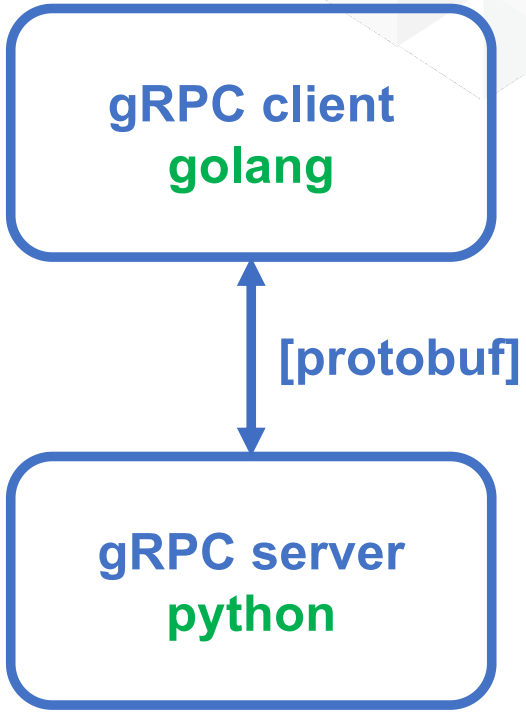
You can use the `sorted` function in Python to order an array of tuples based on the second item in each tuple. Here's an example function that takes an array of tuples as an argument and returns a new array with the tuples sorted in descending order based on the second item in each tuple:

python Copy code

```python
def sort_array(arr):
    return sorted(arr, key=lambda x: x[1], reverse=True)
```

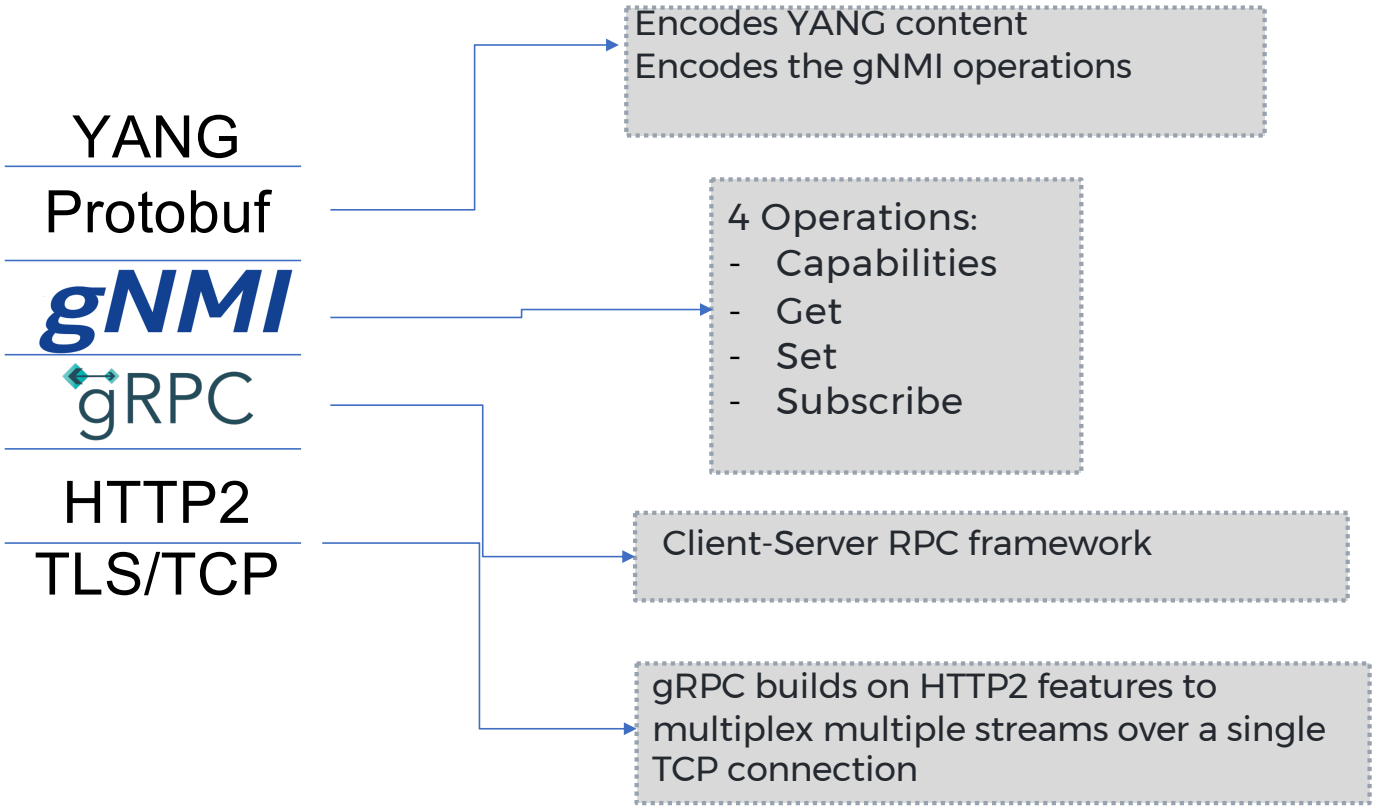
What is GNMI?

- gRPC Network Management Interface
- Open-source protocol developed by Google
- Versatile, efficient, and scalable
  - Protocol buffers
- Retrieve and configure network state information



Source: https://gnmic.kmrd.dev/

gRPC Network Management Interface



In this Tutorial

gNMIc

https://gnmic.kmrd.dev/

Configure (Set)

```
clab-dc-k8s-LEAF-DC-1:
  network-instances:
    - name: l2evpn1001
      admin-state: enable
      type: mac-vrf
      evi: 1001
      vni: 1001
      vxtype: bridged
      anycast-gw: 10.0.1.1/24
    - name: l2evpn1002
      admin-state: enable
      type: mac-vrf
```

pygnmi

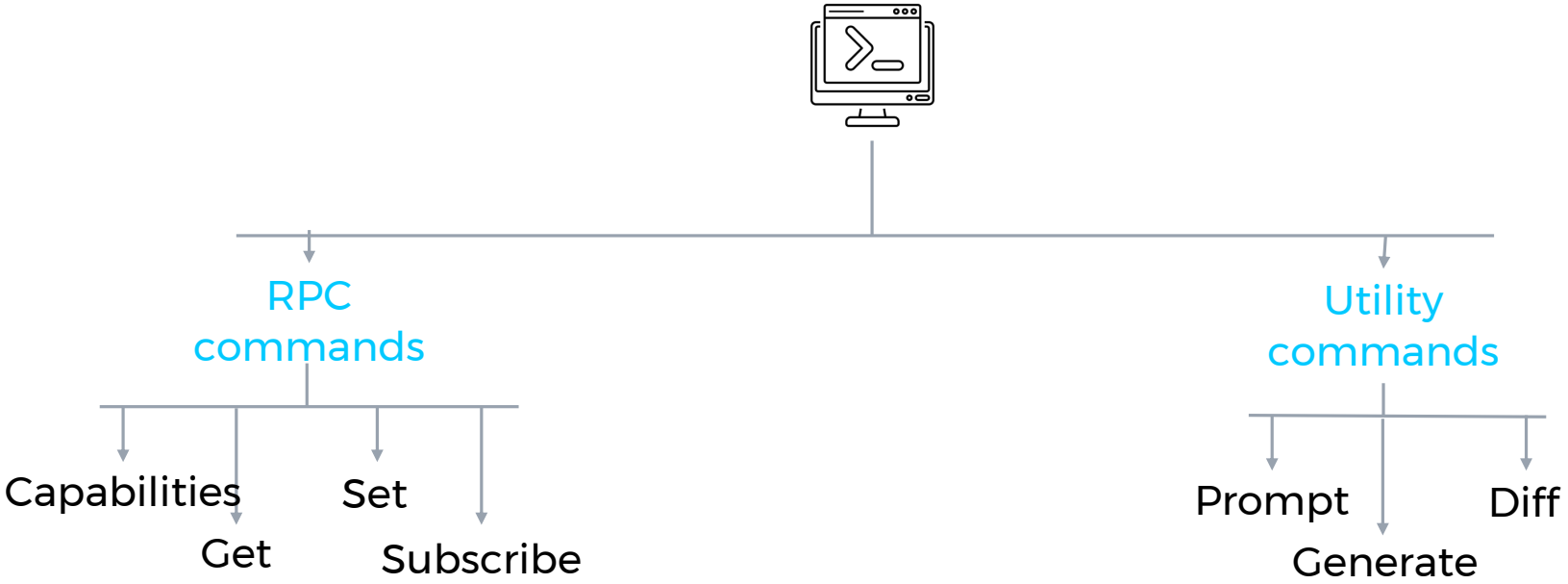
https://github.com/akarneliuk/pygnmi

Extract (Get)

```
username: admin
password: admin
port: 830
gnmi_port: 57400
insecure: True
skip_verify: True
hostkey_verify: False
switches:
  srl:
    clab-dc-k8s-LEAF-DC-1: None
    clab-dc-k8s-LEAF-DC-2: None
    clab-dc-k8s-BORDER-DC: None
```

gNMIc

Rich CLI to explore and test gNMI enabled targets

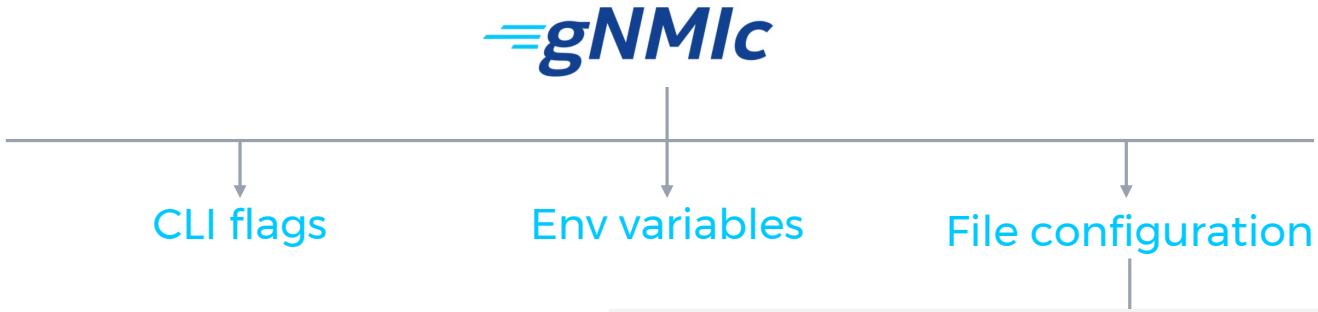


Source: https://gnmic.kmrd.dev/

gNMIc

Configure the way you want

Source: https://gnmic.kmrd.dev/




```
$ cat .gnmic.yaml

address: router1
username: admin
password: admin
insecure: true
encoding: json_ietf
get-path:
  - /interfaces/interface[name=mgmt0]

$ gnmic --config .gnmic.yaml get
```

pyGNMI

https://github.com/akarneliuk/pygnmi

Python library using the GNMI protocol.

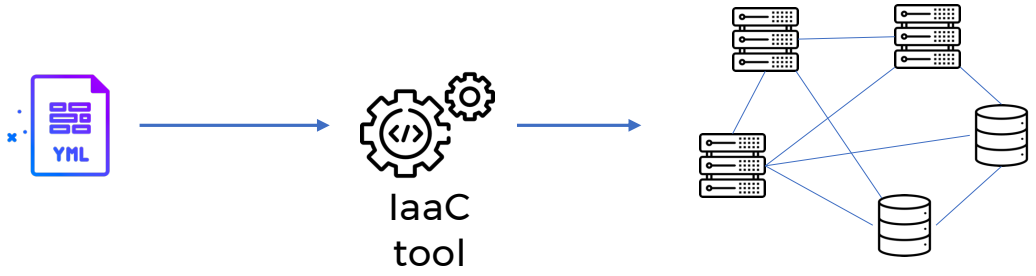
Some benefits over other tools include:

- Cross-platform support.
- Ease of use.
- Scalability.
- Customization
- Security (TLS).

Advanced Troubleshooting Tasks

Clab: Bringing declarativeness to networking labs

IT

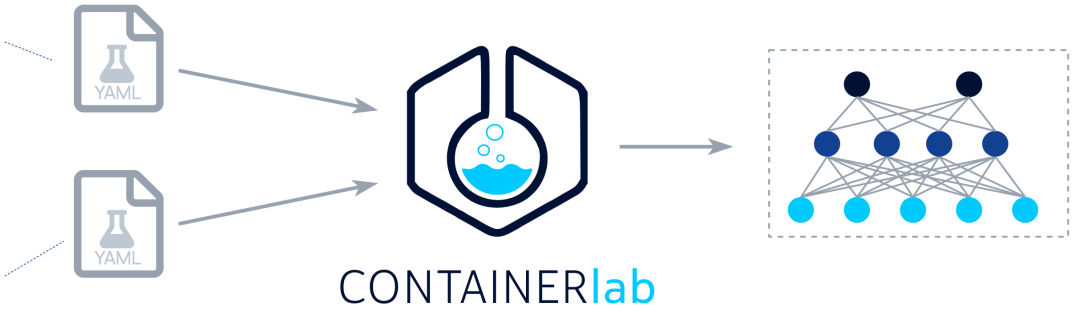


Network Labs

```
name: mylab

topology:
  nodes:
          :
          :

  links:
    -     :
```



NANOG™

What is EVPN-VXLAN?

- Scalable, flexible, and efficient
  - Easier to manage and scale large networks.
- EVPN = VPN technology that creates Ethernet-based virtual networks
  - Initially developed for MPLS networks.
- EVPN uses BGP to distribute MAC and IP address information.
  - Allowing efficient forwarding of traffic between virtualized networks.
- VXLAN (Virtual Extensible LAN) is a tunneling protocol
  - Used to extends Layer 2 networks over Layer 3.

VLANs
Scalability issues
Lack of flexibility

MPLS
Complexity

L3 Fabrics with EVPN

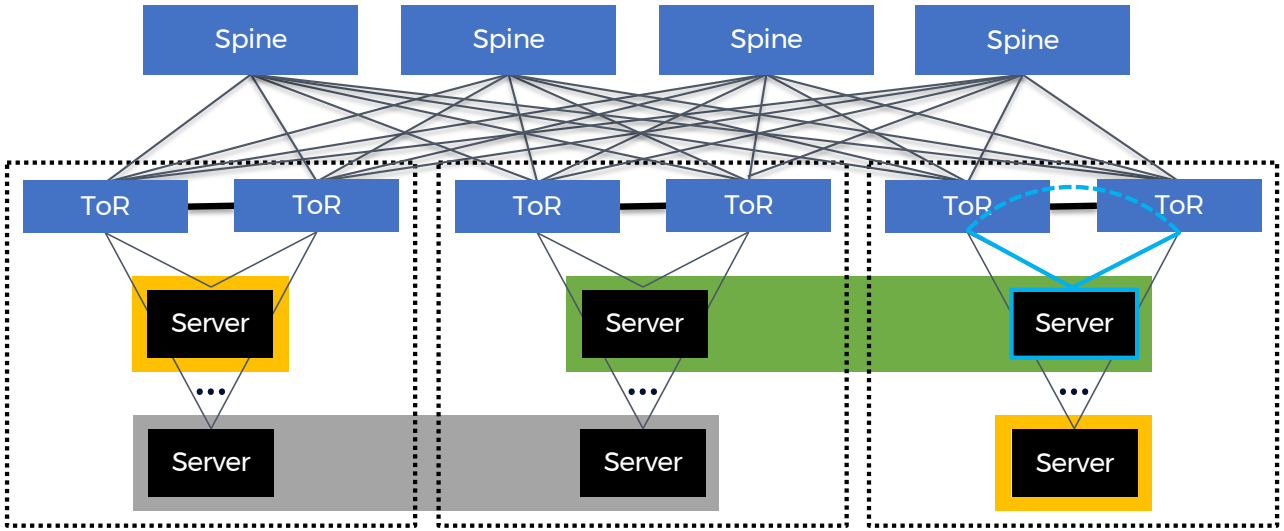
Goals

- Flexible growth
- Better link utilization
- Scalability
- Efficient L2 mobility
- Strong security
- Open multi-homing

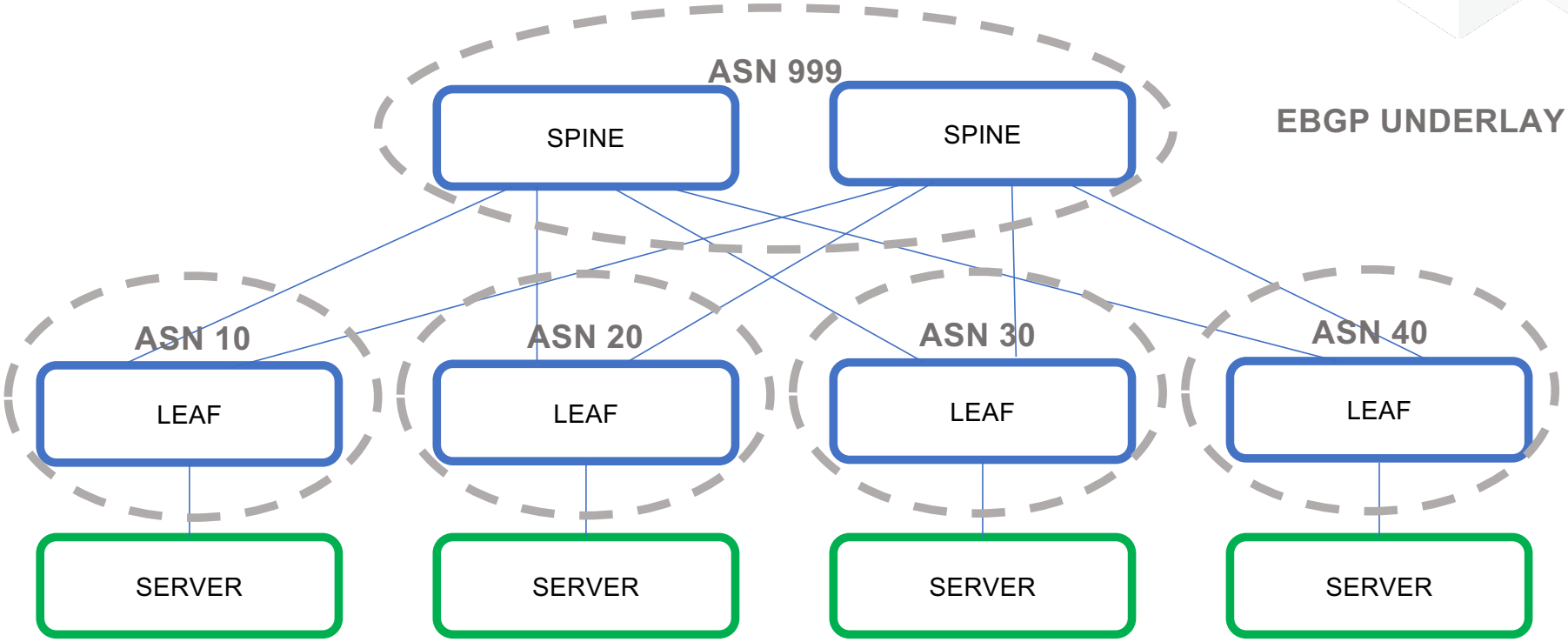
Challenges

- Troubleshooting
- Learning curve

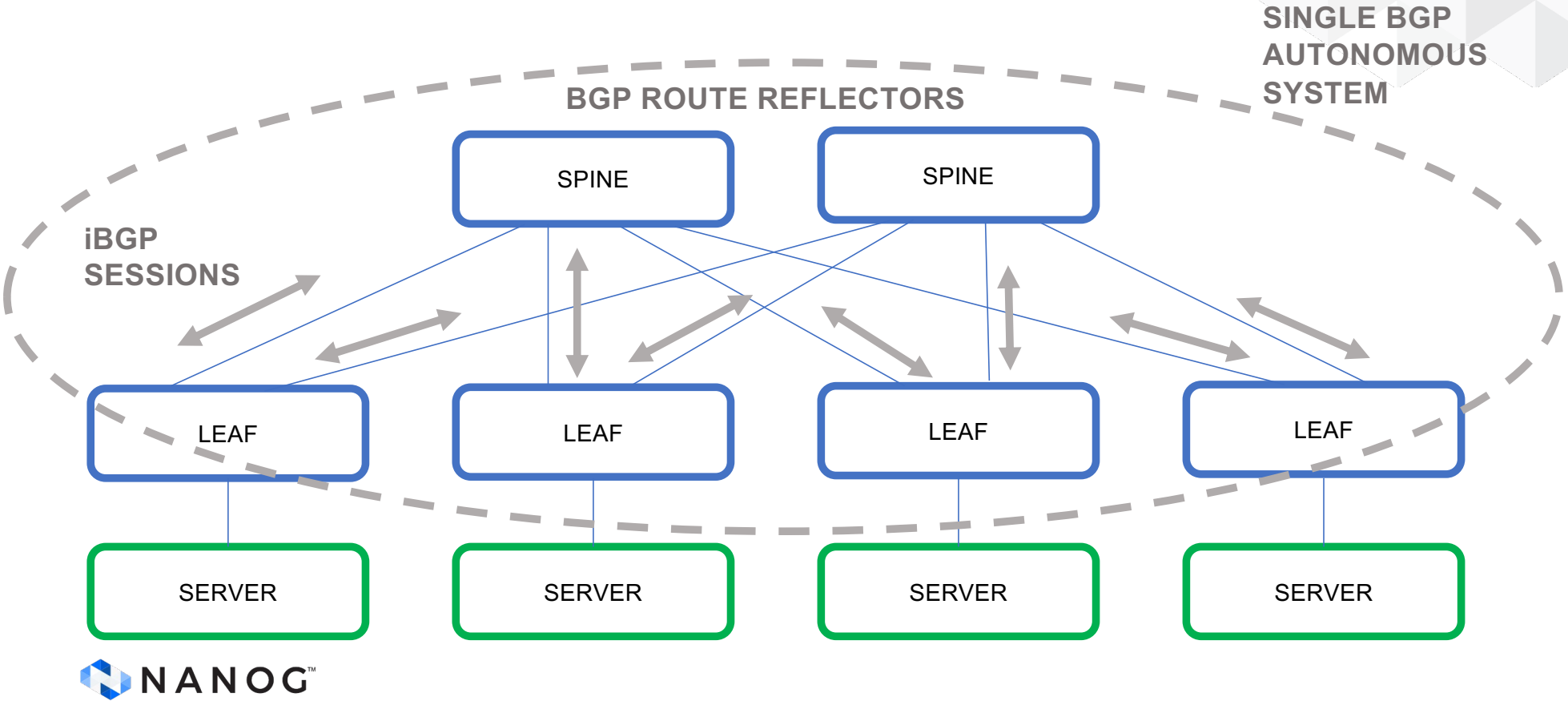
L2 domain(s)



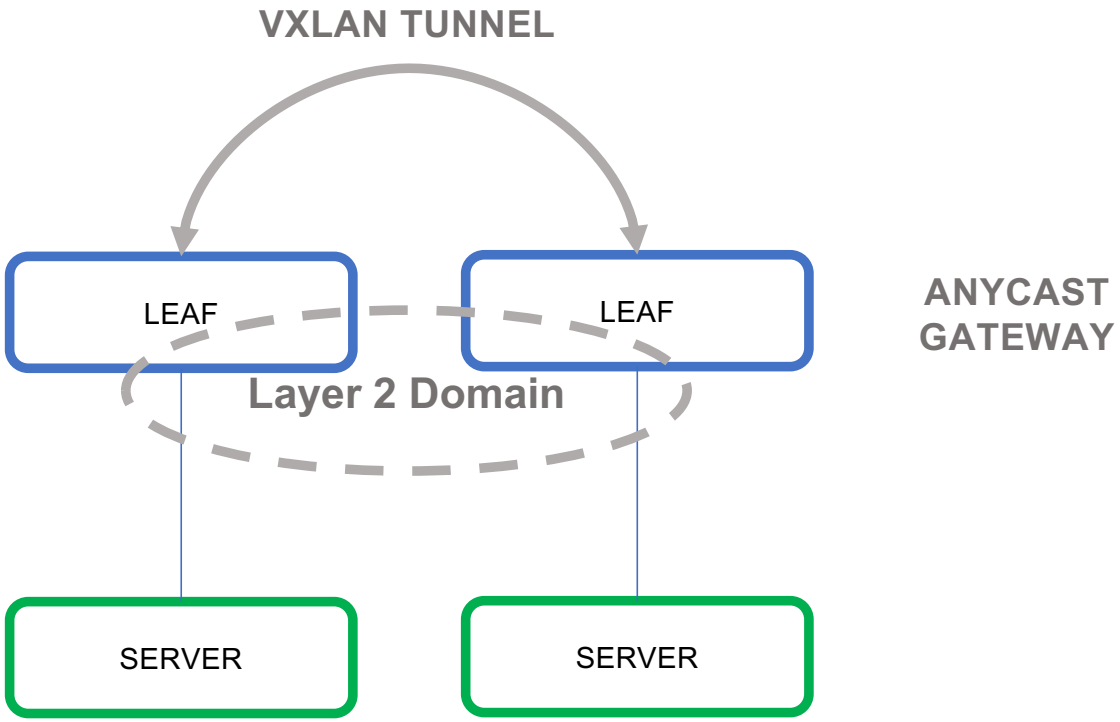
How EVPN-VXLAN works?



How EVPN-VXLAN works?



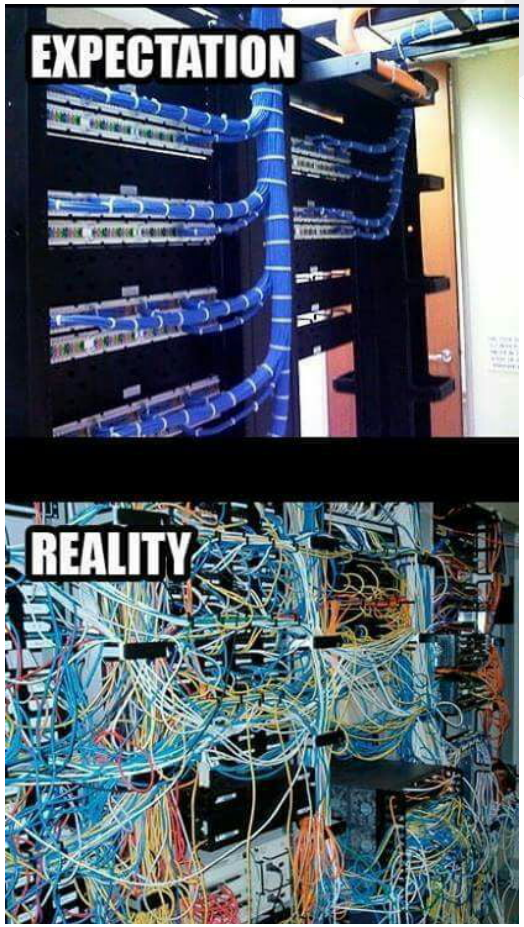
How EVPN-VXLAN works?



Troubleshoot a DC Fabric using EVPN-VXLAN.

All Switches

- Is BGP or EVPN address family correctly configured?
- Verify the Route Targets (RTs) and Route Distinguishers (RDs)
- VLAN to VXLAN mappings
- BGP EVPN neighbors are established and exchanging routes



NANOG™

Containerlab: Installation

Installation commands for Fedora33

```
# Install docker
sudo dnf -y install docker
sudo systemctl start docker
sudo systemctl enable docker

# Install containerlab
bash -c "$(curl -sL https://get.containerlab.dev)" -- -v 0.25.1
```

https://containerlab.dev/install/

Git repo for this tutorial

https://github.com/cloud-native-everything/pygnmi-srl-nanog88

```
[~]# git clone https://github.com/cloud-native-everything/pygnmi-srl-nanog88
Cloning into 'pygnmi-srl-apps'...
remote: Enumerating objects: 251, done.
remote: Counting objects: 100% (251/251), done.
remote: Compressing objects: 100% (154/154), done.
remote: Total 251 (delta 54), reused 251 (delta 54), pack-reused 0
Receiving objects: 100% (251/251), 9.94 MiB | 4.74 MiB/s, done.
Resolving deltas: 100% (54/54), done.
```

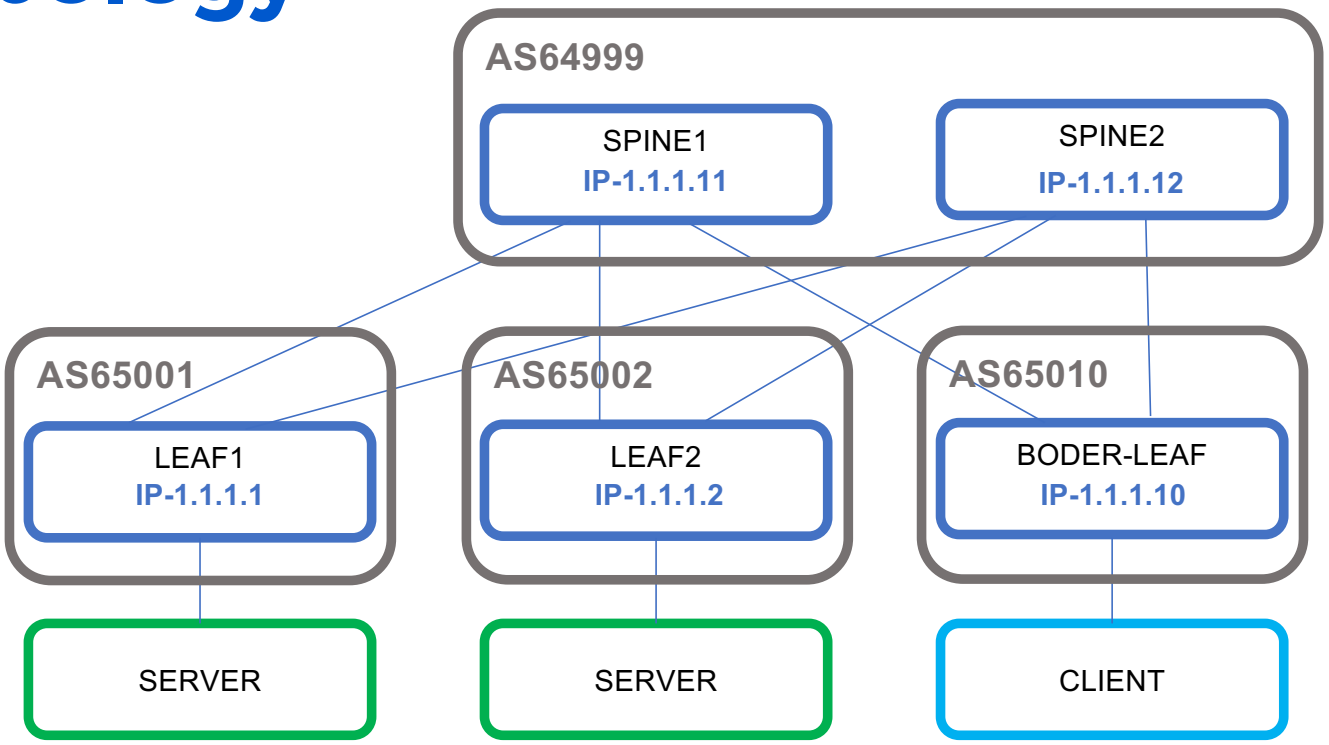
Creating the lab

```
[pygnmi-srl-apps]# clab deploy -t topo.yml
INFO[0000] Containerlab v0.25.1 started
INFO[0000] Parsing & checking topology file: topo.yml
INFO[0000] Creating lab directory: /root/pygnmi-srl-apps/clab-dc-k8s
INFO[0000] Creating docker network: Name="kind", IPv4Subnet="172.18.100.0/16",
IPv6Subnet="", MTU="1500"
INFO[0000] Creating container: "grafana"
INFO[0000] Creating container: "SPINE-DC-2"
INFO[0000] Creating container: "prometheus"
INFO[0000] Creating container: "LEAF-DC-1"
INFO[0000] Creating container: "BORDER-DC"
INFO[0000] Creating container: "SPINE-DC-1"
INFO[0000] Creating container: "LEAF-DC-2"
```

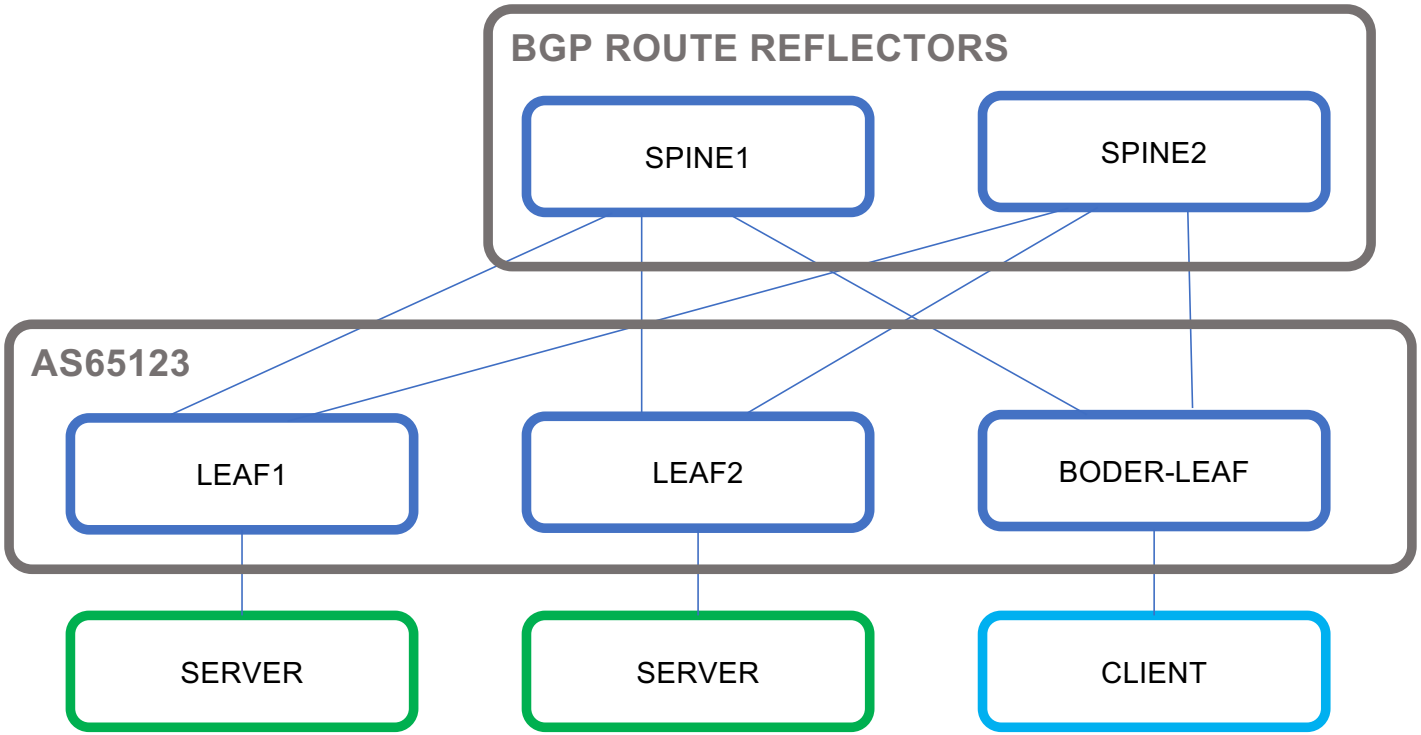
Inspecting the lab

```
[root@rbc-r2-hpe4 pygnmi-srl-apps]# clab inspect -t topo.yml
INFO[0000] Parsing & checking topology file: topo.yml
+---+------------------------+--------------+-------------------------------+-------+-------------------+
| # |          Name          | Container ID |             Image             | Kind  |   IPv4 Address    |
+---+------------------------+--------------+-------------------------------+-------+-------------------+
| 1 | clab-dc-k8s-BORDER-DC  | 9876f09a5580 | ghcr.io/nokia/srlinux:21.6.4 | srl   | 172.18.100.125/16 |
| 2 | clab-dc-k8s-LEAF-DC-1  | 830369bb4d39 | ghcr.io/nokia/srlinux:21.6.4 | srl   | 172.18.100.121/16 |
| 3 | clab-dc-k8s-LEAF-DC-2  | 05d303e50816 | ghcr.io/nokia/srlinux:21.6.4 | srl   | 172.18.100.122/16 |
| 4 | clab-dc-k8s-SPINE-DC-1 | 574ff19416fb | ghcr.io/nokia/srlinux:21.6.4 | srl   | 172.18.100.123/16 |
| 5 | clab-dc-k8s-SPINE-DC-2 | e44d29973290 | ghcr.io/nokia/srlinux:21.6.4 | srl   | 172.18.100.124/16 |
| 6 | clab-dc-k8s-grafana    | e6d5221fa472 | grafana/grafana:latest        | linux | 172.18.100.116/16 |
| 7 | clab-dc-k8s-prometheus | 533473420ff1 | prom/prometheus:latest        | linux | 172.18.100.115/16 |
+---+------------------------+--------------+-------------------------------+-------+-------------------+
```

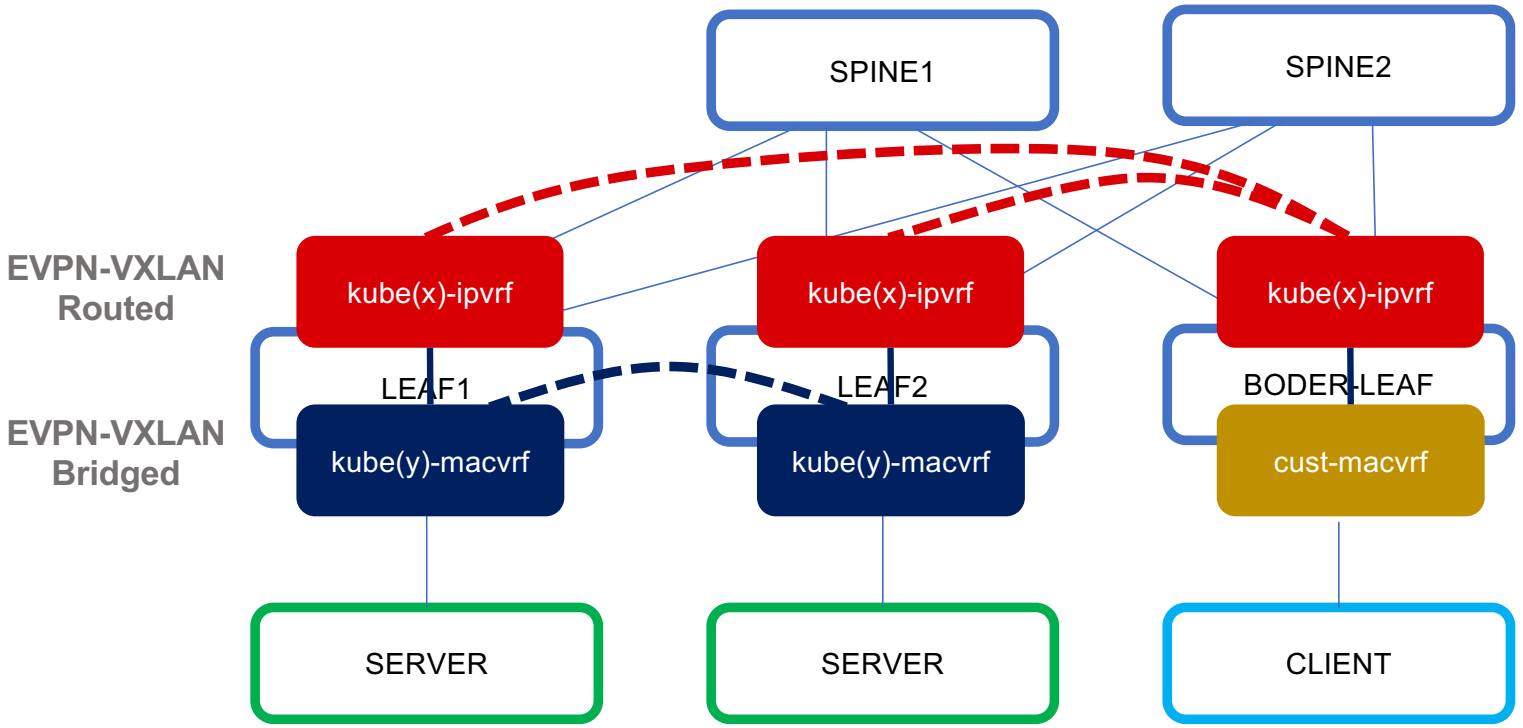
Lab Topology: eBGP Underlay Topology



Lab Topology: iBGP EVPN Overlay



Lab Topology: iBGP EVPN Overlay



Installing pyGNMI requirements

```
[root@rbc-r2-hpe4 network-instance]# pip install pygnmi
WARNING: Running pip install with root privileges is generally not a good idea. Try `pip
install --user` instead.
Collecting pygnmi
  Downloading pygnmi-0.8.9.tar.gz (30 kB)
Collecting grpcio
  Downloading grpcio-1.54.0-cp39-cp39-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (5.1 MB)
     |████████████████████████████████| 5.1 MB 31.3 MB/s
Collecting protobuf
  Downloading protobuf-4.22.3-cp37-abi3-manylinux2014_x86_64.whl (302 kB)
     |████████████████████████████████| 302 kB 35.1 MB/s
Collecting cryptography
  Downloading cryptography-40.0.2-cp36-abi3-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
(3.7 MB)
     |████████████████████████████████| 3.7 MB 38.0 MB/s
Collecting dictdiffer
  Downloading dictdiffer-0.9.0-py2.py3-none-any.whl (16 kB)
```

Installing pyGNMI requirements

```
[root@rbc-r2-hpe4 py-scripts]# pip install pyyaml
WARNING: Running pip install with root privileges is generally not a good idea. Try `pip install --user` instead.
Collecting pyyaml
  Downloading PyYAML-6.0-cp39-cp39-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_12_x86_64.manylinux2010_x86_64.whl (661 kB)
     |████████████████████████████████| 661 kB 15.7 MB/s
Installing collected packages: pyyaml
Successfully installed pyyaml-6.0
```

```
[root@rbc-r2-hpe4 py-scripts]# pip install prettytable
WARNING: Running pip install with root privileges is generally not a good idea. Try `pip install --user` instead.
Collecting prettytable
  Downloading prettytable-3.7.0-py3-none-any.whl (27 kB)
Collecting wcwidth
  Downloading wcwidth-0.2.6-py2.py3-none-any.whl (29 kB)
Installing collected packages: wcwidth, prettytable
```

```
[root@rbc-r2-hpe4 py-scripts]# pip install tabulate
WARNING: Running pip install with root privileges is generally not a good idea. Try `pip install --user` instead.
Requirement already satisfied: tabulate in /usr/local/lib/python3.9/site-packages (0.9.0)
```

Installing GNMIc

```
bash -c "$(curl -sL https://get-gnmic.kmrd.dev)”
```

```
gnmic -a 172.18.100.122:57400 -u admin -p admin --skip-verify get -e
json_ietf --path /system/name/host-name
```

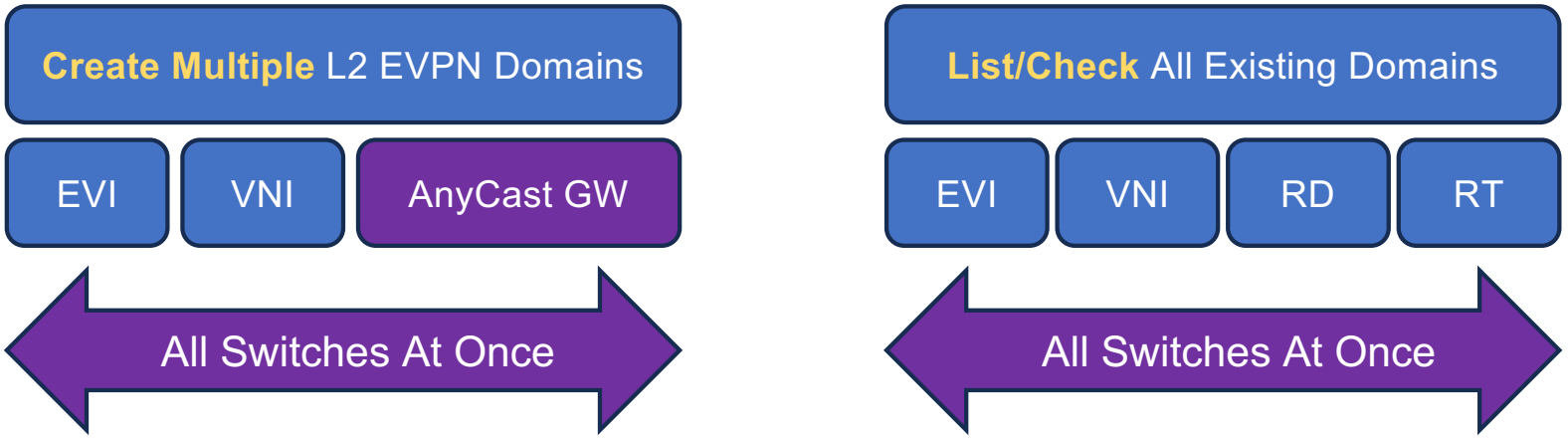
In this Tutorial

GO gNMIc

Configure (Set)

python™ pyGNMI

Extract (Get)



Scripting with ChatGPT

- Functions and Classes.
  - Easy to edit and simplify chatgpt interactions
- Keep using the same chat thread
  - Keeping context for chatgpt brings better answers.
- Clear questions, complete information, better answers
- Don’t use too much code as input

Scripting with ChatGPT

Put output in an array using pygnmi function: “`from pygnmi.client import gNMIclient import yaml...`

Show how will you do a helper method for functions a() and b()?

Is there table function to import in python to make this table more dynamic

NANOG™

Scripting with ChatGPT

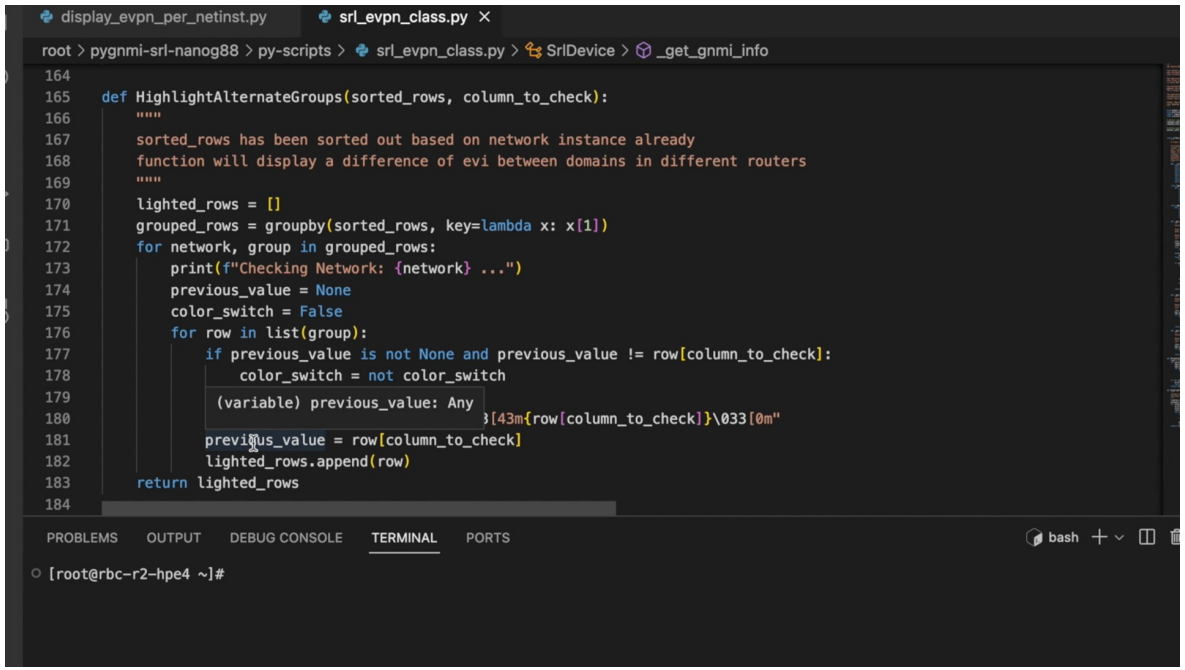
Show me previous table creation as a function

Remove "ssl_target_name_override is applied, should be used for testing only!" this from the output

Error handling here: “
```
def main():
    with open(args.filename, 'r') as fh: …
```

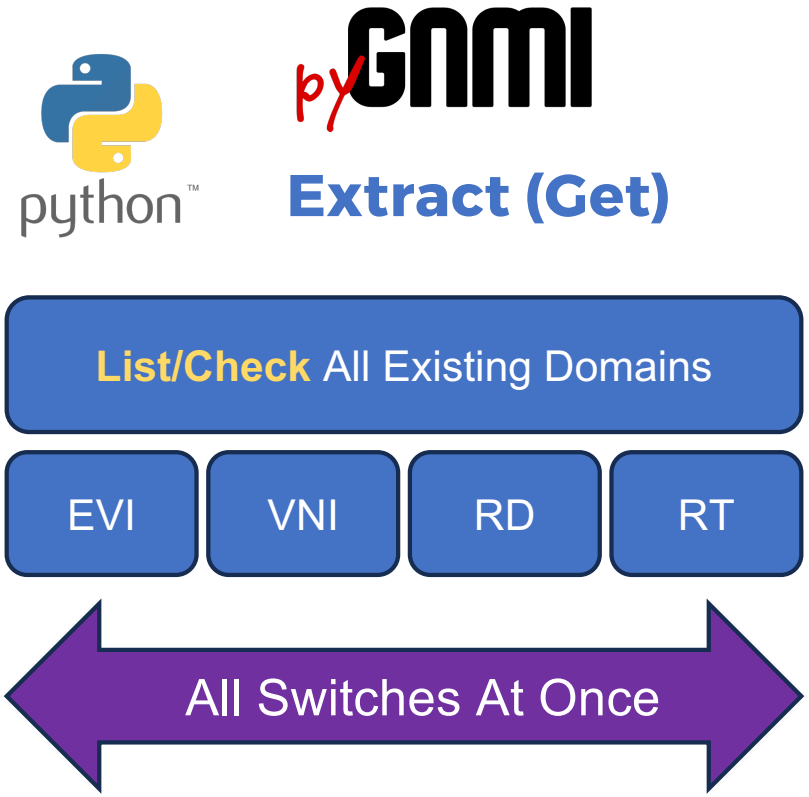
NANOG™

Scripting with ChatGPT



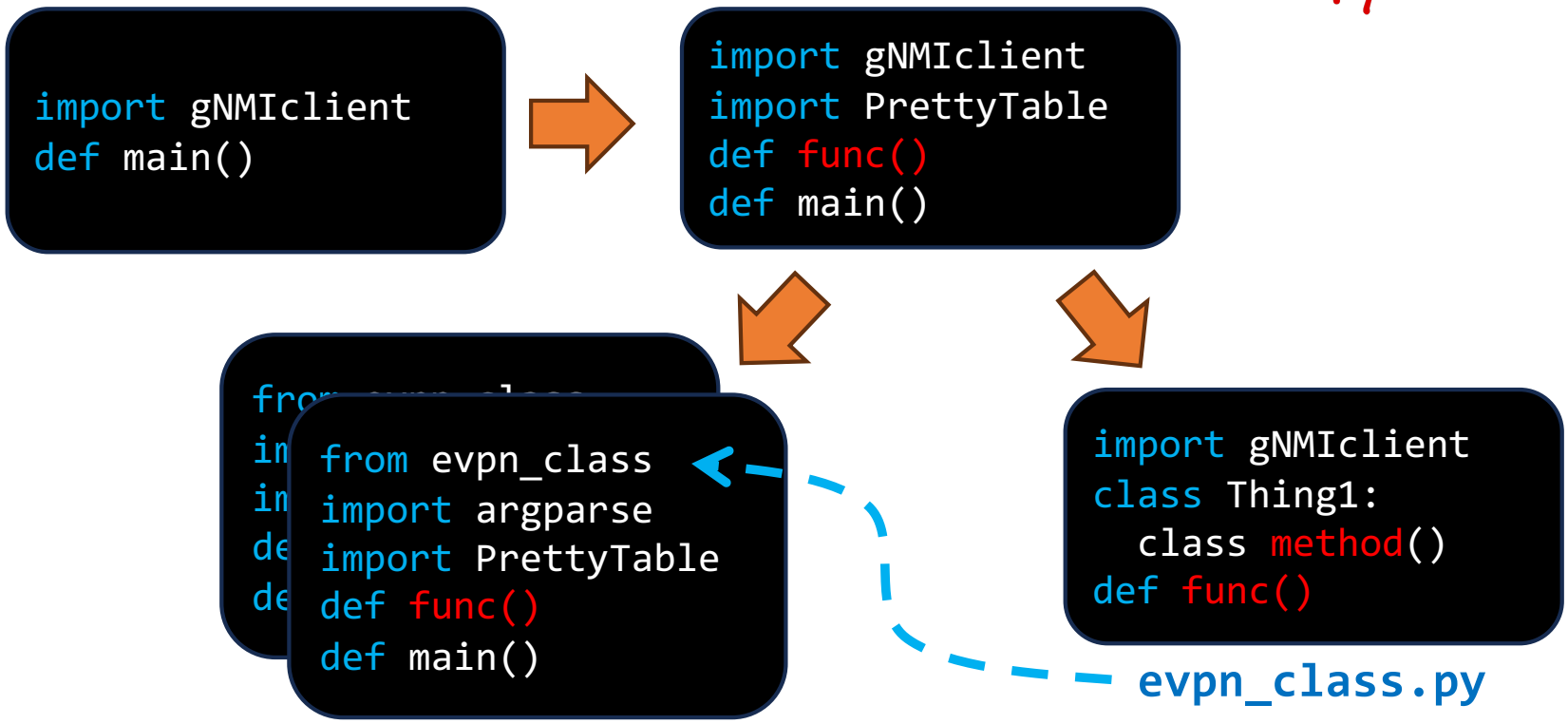
ChatGPT

NANOG™



- List All VRFs in All Switches sorted by **Switch Name**
- List All VRFs in All Switches sorted by **Network Instance**
  - Identify any misconfiguration in **EVI**

Scripting: My Journey



display_evpn_per_router.py
display_evpn_per_netinst.py

Python Class

- Modularity
- Reusability
- Abstraction

```
class Rectangle:
    def __init__(self, width, height):
        self.width = width
        self.height = height

    def area(self):
        return self.width * self.height

    def perimeter(self):
        return 2 * (self.width + self.height)
```

NANOG™

pyGNMI

Python Class

```
class SrlDevice:
def __init__(self, router, port, model, release, username,
               password, skip_verify=DEFAULT_SKIP_VERIFY):
        self.router = router
        self.port = port
        self.password = password
        self.username = username
        self.skip_verify = skip_verify
        self.model = model
        self.release = release
        self.bgp_evpn = self.get_bgp_evpn_info()
        self.bgp_vpn = self.get_bgp_vpn_info()
```

```
 srl_devices = []
    for router in routers:
        srl_devices.append(SrlDevice(router, port, DEFAULT_MODEL,
                                     DEFAULT_RELEASE, username, password, skip_verify))
```

Helper Function

```
def _get_gnmi_info(self, gnmi_path):
        info = []
        result = None
        try:
            with gNMIclient(target=(self.router, self.port), username=self.username,
                            password=self.password, skip_verify=True) as gc:
                result = gc.get(path=gnmi_path)
        except Exception as e:
            print(f"Failed to connect to router or fetch data: {e}")
```

YAML: Scripting

- YAML file helps to parametrize the script

```yaml
---
username: admin
password: admin
port: 830
gnmi_port: 57400
insecure: True
skip_verify: True
hostkey_verify: False
switches:
  srl:
    clab-dc-k8s-LEAF-DC-1: None
    clab-dc-k8s-LEAF-DC-2: None
    clab-dc-k8s-BORDER-DC: None
```

Checking L2/L3 EVPN Domains

Extract (Get)

List/Check All Existing Domains

EVI | VNI | RD | RT

All Switches At Once

NANOG™

pyGNMI

Checking misconfigurations

Network instance	ID	EVPN Admin state	VXLAN interface	EVI	ECMP	O
kube-ipvrf	1	enable	vxlan1.4	4	4	
kube-ipvrf	1	enable	vxlan1.4	4	4	
kube-ipvrf	1	enable	vxlan1.4	4	4	
kube_macvrf	1	enable	vxlan1.1	1	1	
kube_macvrf	1	enable	vxlan1.1	1	1	
l2evpn1001	2	enable	vxlan2.1001	1001	1	
l2evpn1001	2	enable	vxlan2.1001	1001	1	
l2evpn1002	2	enable	vxlan2.1002	1002	1	
l2evpn1002	2	enable	vxlan2.1002	1002	1	
l2evpn1003	2	enable	vxlan2.1003	1003	1	
l2evpn1003	2	enable	vxlan2.1003	1003	1	
l2evpn1004	2	enable	vxlan2.1004	1004	1	
l2evpn1004	2	enable	vxlan2.1004	1004	1	
l2evpn1005	2	enable	vxlan2.1005	1015	1	
l2evpn1005	2	enable	vxlan2.1005	1005	1	
l2evpn1006	2	enable	vxlan2.1006	1006	1	
l2evpn1006	2	enable	vxlan2.1006	1006	1	
l3evpn	1	enable	vxlan1.2	2	4	
l3evpn	1	enable	vxlan1.2	2	4	

NANOG™

pyGNMI

Scripting

- Why my table is so beautiful?

```
table = PrettyTable()
table.field_names = ['Router', 'Network instance', 'ID', 'EVPN Admin state', 'VXLAN interface',
                     'EVI', 'ECMP', 'Oper state', 'RD', 'import-rt', 'export-rt']
```

```
sorted_rows = sorted(rows, key=lambda x: x[1])
print("Table: Sorted by Network Instance")
highlighted_rows = HighlightAlternateGroups(sorted_rows, 5)  # Assuming Network Instance is the 1st
table = tabulate(highlighted_rows, headers=['Router', 'Network instance', 'ID', 'EVPN Admin state',
                                            'VXLAN interface', 'EVI', 'ECMP', 'Oper state',
                                            'RD', 'import-rt', 'export-rt'], tablefmt="pretty")
print(table)
```

NANOG™

pyGNMI

Scripting

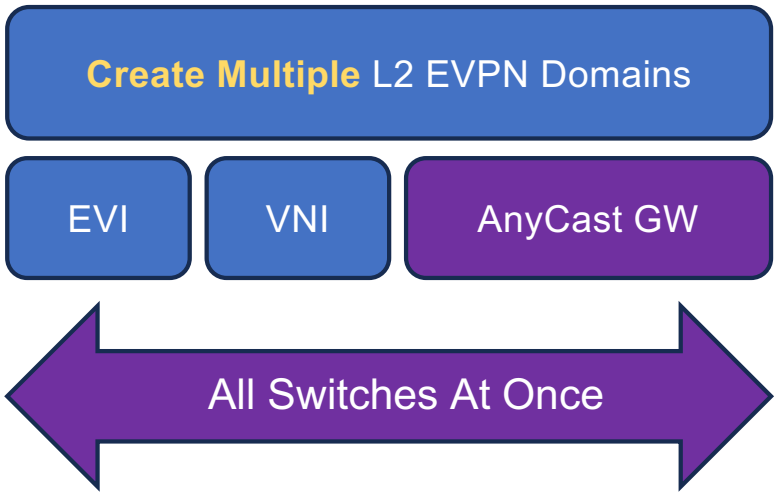
- Reading from YAML file into an array

```python
def main():
    try:
        with open(args.filename, 'r') as fh:
            router_info = yaml.safe_load(fh)
    except FileNotFoundError:
        print(f"File {args.filename} not found.")
        return
    except yaml.YAMLError as exc:
        print(f"Error in configuration file: {exc}")
        return

    try:
        switches = router_info['switches']
        routers = switches['srl']
        username = router_info['username']
        password = router_info['password']
        port = router_info['gnmi_port']
        skip_verify = router_info['skip_verify']
    except KeyError as e:
        print(f"Key {e} not found in configuration file.")
        return
```

gNMIc

Configure (Set)



- Create **Multiple L2 Domains** on multiples Switches at once
- Add **AnyCast GW** to all of them connected to a **Specific L3 VRF**.

GO Templates

```
replaces:
{{ $target := index .Vars .TargetName }}
{{- range $netinstances := index $target "network-instances" }}
  - path: "/network-instance[name={{ index $netinstances "name" }}]"
    encoding: "json_ietf"
    value:
      admin-state: enable
      type: {{ index $netinstances "type" | default "mac-vrf" }}
      description: {{ index $netinstances "description" | default "whatever" }}
      vxlan-interface:
        - name: vxlan2.{{ index $netinstances "vni" }}
{{- end }}
```

- Flexibility and Reusability
- Dynamic Content Generation
- Separation of Logic and Data
- Complex Formatting

NANOG™

gNMIc

Input Var File

- **Readability** - Simple syntax and Structure
- **Structure** - Lists, maps, and nested elements
- **Portability** – Language Agnostic
- **Integration** – Ansible, Kubernetes, Docker

```yaml
clab-dc-k8s-LEAF-DC-1:
  network-instances:
    - name: l2evpn1001
      admin-state: enable
      type: mac-vrf
      evi: 1001
      vni: 1001
      vxtype: bridged
      anycast-gw: 10.0.1.1/24
    - name: l2evpn1002
      admin-state: enable
      type: mac-vrf
      evi: 1002
      vni: 1002
      vxtype: bridged
      anycast-gw: 10.0.2.1/24
```

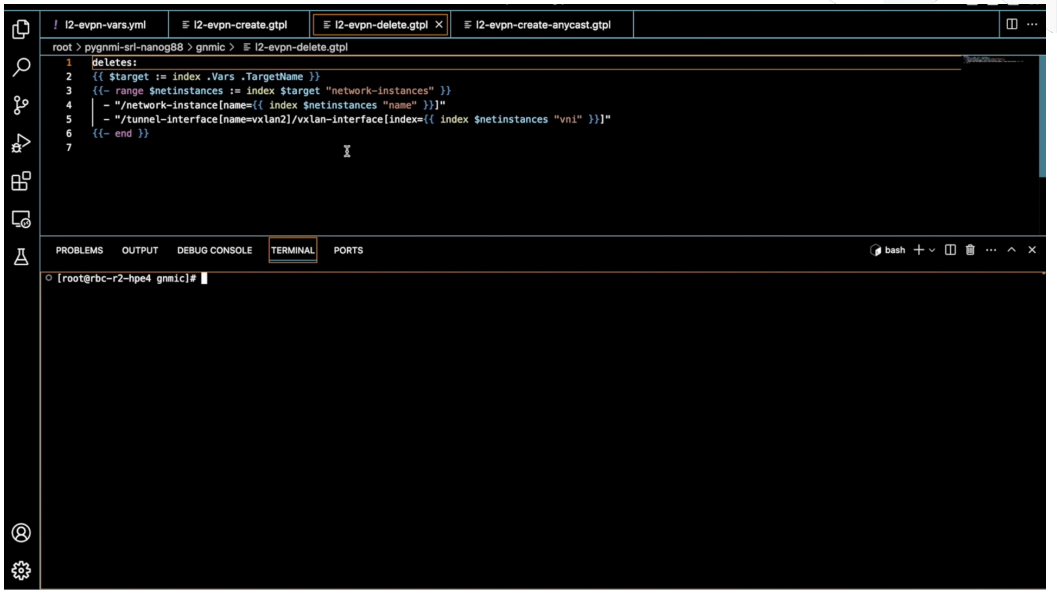
GNMIc replace/update video

Configure (Set)

Create Multiple L2 EVPN Domains

EVI | VNI | AnyCast GW

All Switches At Once



NANOG™

gNMIc

Final Words

In this session, we covered the following topics:

- Creating a network lab using Containerlab
- Exploring the Spine/Leaf + Border-Leaf architecture.
- Understanding Layer 2/3 EVPN-VXLAN and its configuration.
- Configuring multiple devices effortlessly with GNMIc and Go Templates.
- Troubleshooting EVPN-VXLAN designs using pyGNMI scripting for effective network analysis.

Additional resources

- ChatGPT and Networking Engineering - Mike Starr
  - Potential use cases for ChatGPT in network engineering
  - https://youtu.be/stzPJspkUUs
- Containerlab - running networking labs with Docker UX – Roman and Karim
  - https://youtu.be/qigCla1qY3k
- gNMIc - an intuitive gNMI CLI and a feature-rich telemetry collector - Karim
  - https://youtu.be/v3CL2vrGD_8

Thanks!

Mau Rojas
bio.site/pinrojas

NANOG™

Additional Slides

Protocol Buffers Serialization

- Serialization in Protocol Buffers refers to the process of converting structured data objects into a compact and efficient binary format.
- Protocol Buffers allows you to define a schema for your data using a .proto file, and then use a library to create instances of that message in your code.
- Once you have a message object, you can serialize it to a binary format that is smaller and faster to transmit than text-based formats like JSON or XML.
- Protocol Buffers' binary format also supports schema evolution, so you can modify your message structure over time while maintaining backward compatibility.
- In summary, serialization in Protocol Buffers is the process of converting structured data into a binary format that is efficient for transmission and storage, and supports schema evolution.

Protocol Buffers Example

Assuming you have a .proto file that defines your message structure, you can use the protoc command-line tool to generate Python code for your message. For example, suppose you have a person.proto file that defines a Person message:

```
syntax = "proto3";

message Person {
  string name = 1;
  int32 age = 2;
}
```

You can generate Python code for this message by running the following command:

```
$ protoc --python_out=.
person.proto
```

This generates a person_pb2.py file, which contains Python classes that correspond to your message fields. You can import this module into your Python code and use it to create and serialize instances of your message.

Protocol Buffers Example

Here's an example:

In this example, we first import the person_pb2 module, which contains the Python classes generated by the protoc command. We then create a new Person instance, set its fields, and serialize it to bytes using the SerializeToString() method. We then deserialize the message using the ParseFromString method, which populates a new Person instance with the serialized data. Finally, we access the fields of the deserialized message using Python attribute access.

```
import person_pb2

# Create a new Person instance and set its
fields
person = person_pb2.Person()
person.name = "Alice"
person.age = 30

# Serialize the message to bytes
data = person.SerializeToString()

# Deserialize the message from bytes
new_person = person_pb2.Person()
new_person.ParseFromString(data)

# Access the fields of the deserialized
message
print(new_person.name)  # "Alice"
print(new_person.age)  # 30
```

Troubleshoot a DC Fabric using EVPEN-VXLAN.

```
4. Validate the EVPN MAC and IP routes:
=======================================
       - Check that the EVPN Type-2 and Type-5 routes are being advertised and
       received by the appropriate devices.
       - Confirm that the MAC and IP address bindings are correct in the EVPN
       database.
       - Inspect the MAC and ARP tables to ensure that the correct MAC and IP address
       entries are present.
5. Test end-to-end connectivity:
================================
       - Perform ping tests between endpoints in the same and different VXLAN
       segments.
       - Use traceroute to check the path taken by the traffic between endpoints.
       - If possible, test application-level connectivity between endpoints to
       validate proper network operation.
```