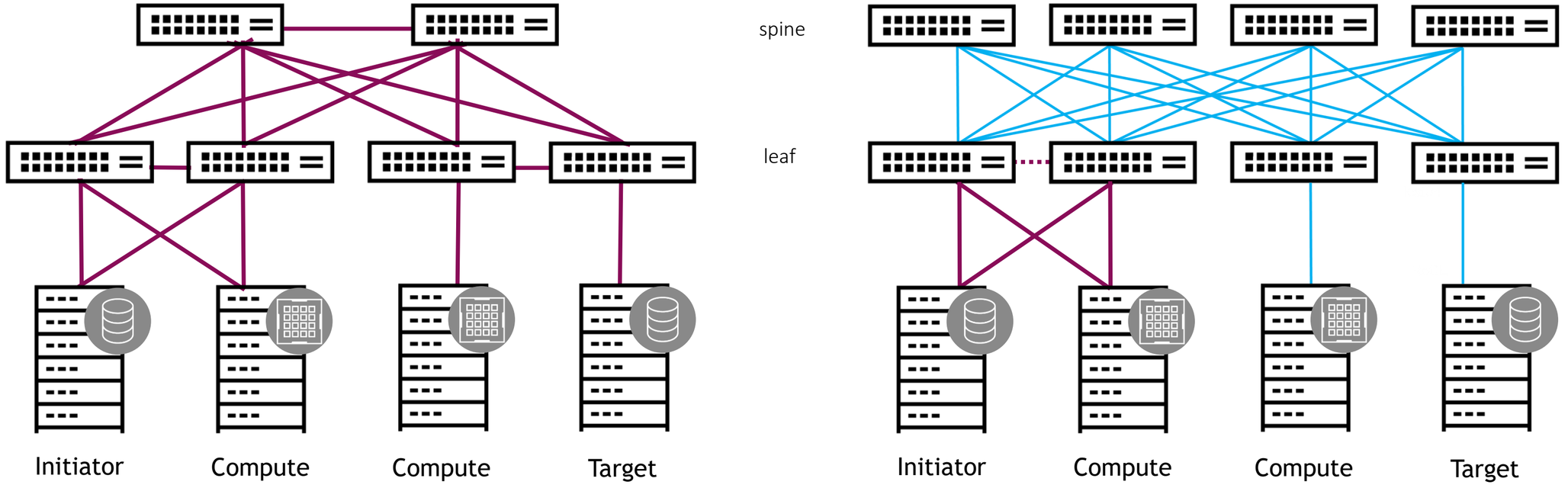
5 MIN TO INFRASTRUCTURE

ANDREAS LA QUIANTE & TOMER LIOR, 09-NOVEMBER-2021

MOTIVATION

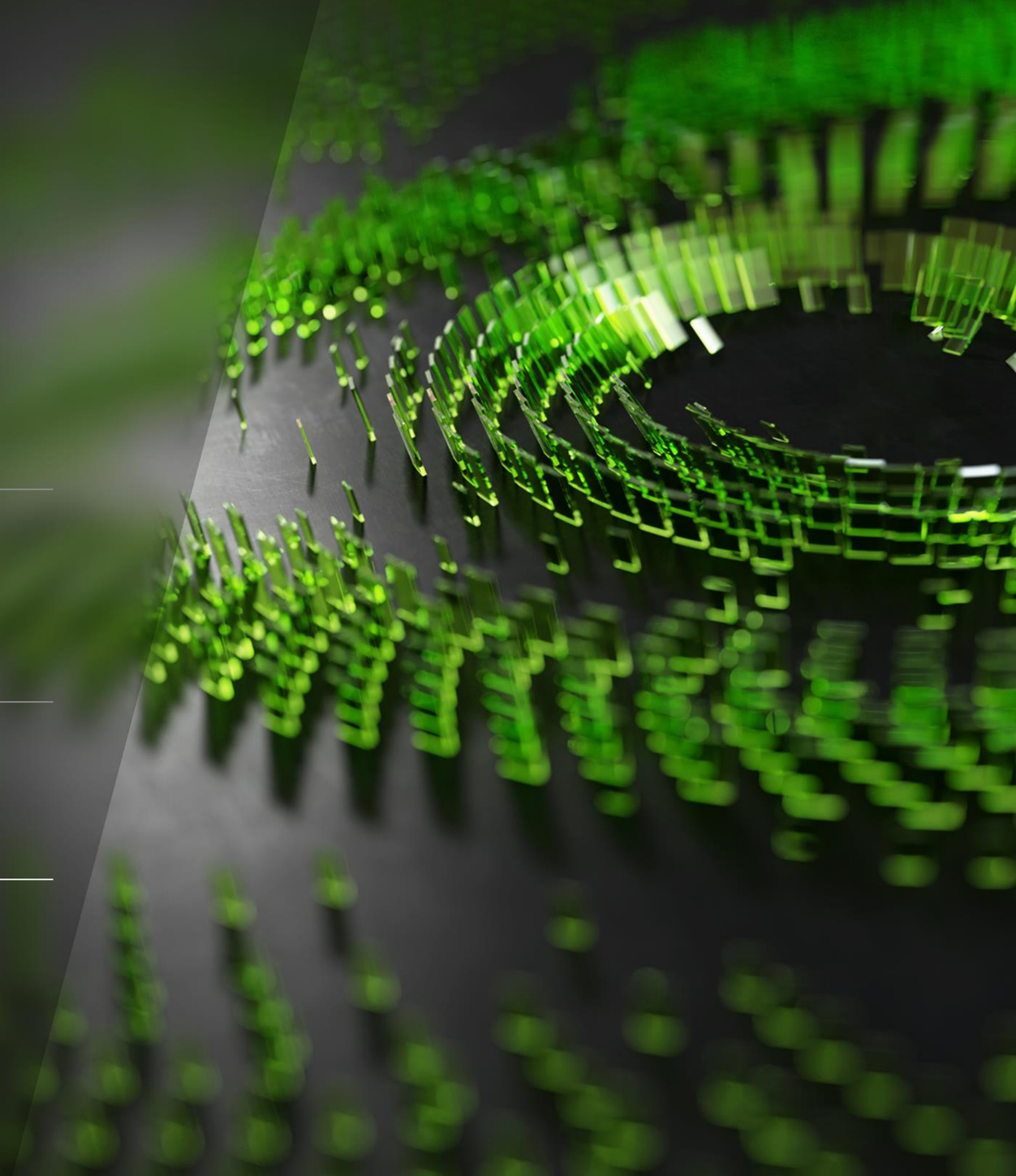
Data Center design is transitioning from Layer 2 to Layer 3
for enterprises with overlays (EVPN)
and includes automation



ALQ

AGENDA

Orchestration & SVC Ansible and Git	1h
Linux Networking Lab layout Cumulus Linux	4h



ONCE UPON A TIME

A rich lady went to a famous painter asking for a special gift, a drawing of the favorite animal of/for her husband. The painter agreed to create a unique drawing but stated it takes one year…

PREPARATION

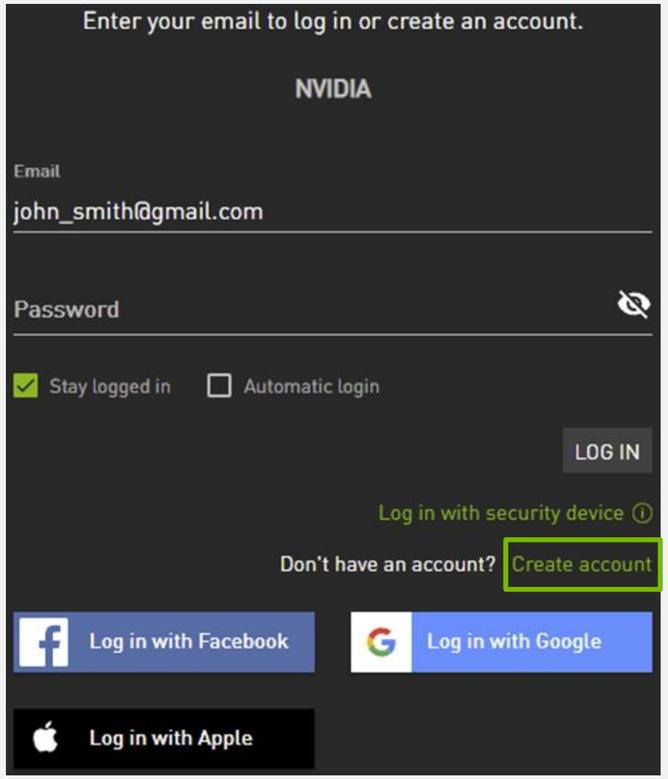
Useful skills but mostly interest

- Networking (Cumulus/FreeRangeRouting)
- Linux (Debian/Ubuntu)
- DevOps (Git/Ansible)
- Programming (Python)

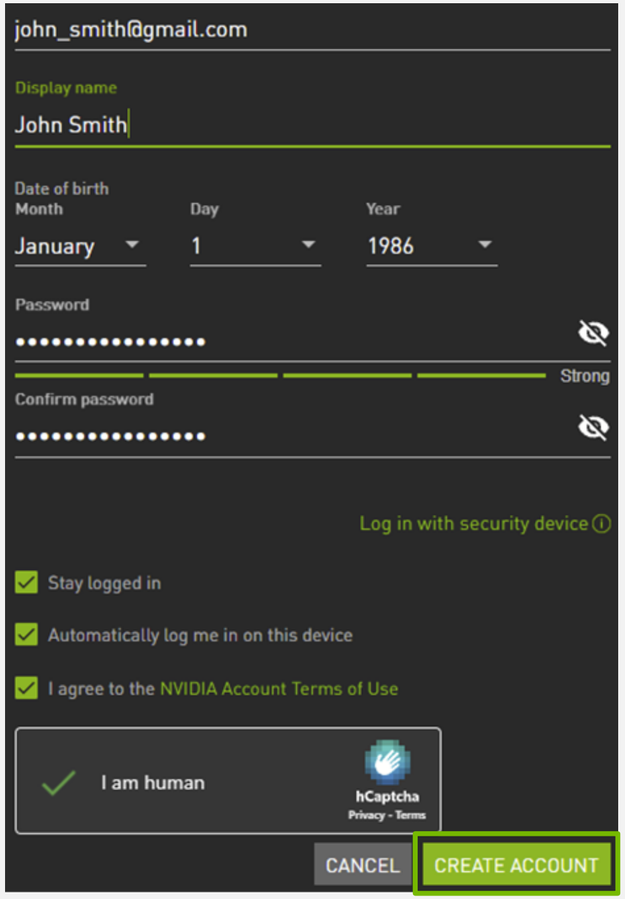
NVIDIA AIR

NVIDIA network simulations platform

1. If you don’t have an account, you’ll be redirected to create one.
   Press on Create account



2. Fill all information.
   Press Create account



3. Verify your email address to finish registration

ALQ


ORCHESTRATION SVC

DISCLAIMER

“This” content is created for teaching concepts not as a blueprint, best-practice or recommendation

- For best-practice please clone the Gitlab repository:
- https://gitlab.com/cumulus-consulting/goldenturtle/cumulus_ansible_modules
- To use a structured/cooky-cutter approach based on roles and templates.

TL

NETWORK AUTOMATION AND ORCHESTRATION

- Automation
  - Setting up a single task to run on its own
- Orchestration
  - Running multiple automated tasks in order to automatically execute a larger workflow or process
- Network Automation
  - The process of automating the configuration, management, testing, deployment, and operations of physical and virtual devices within a network
- Benefits of automation and orchestration:
  - Decreasing IT costs
  - Increasing productivity
  - Standardizing processes so that they are more consistent and reliable

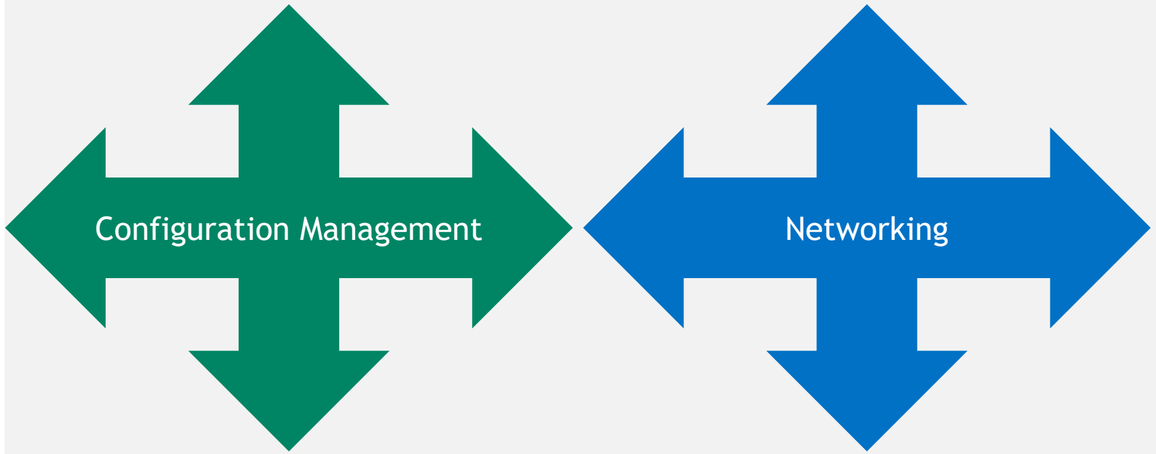
TL

TRIGGER

We Share and Bridge

Still there are questions unanswered and some we touch today during this workshop,
questions like:

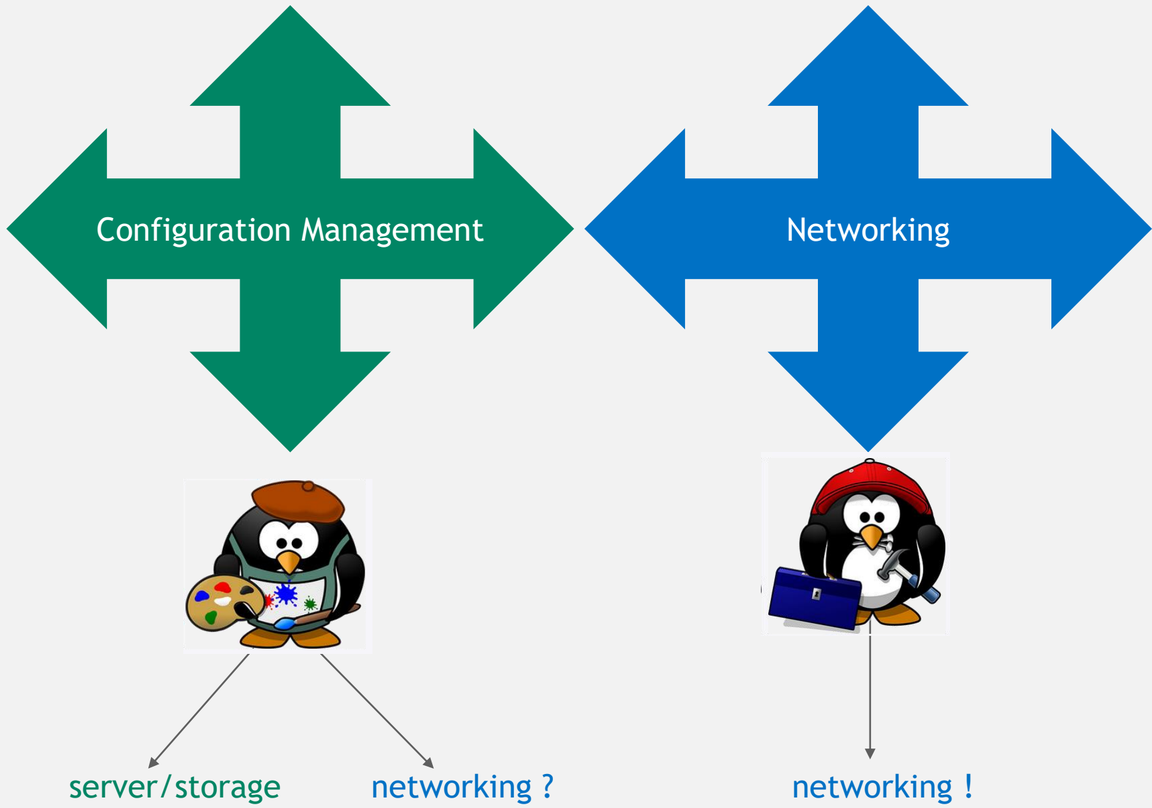
- How to make the dough/面 or can you show some options to create playbooks?
- Where to get the cutters, toppings or which instructions do I need for networking?



TWO TRACKS IN PARALLEL

Building Ansible Playbooks and Building Code-as-Infrastructure

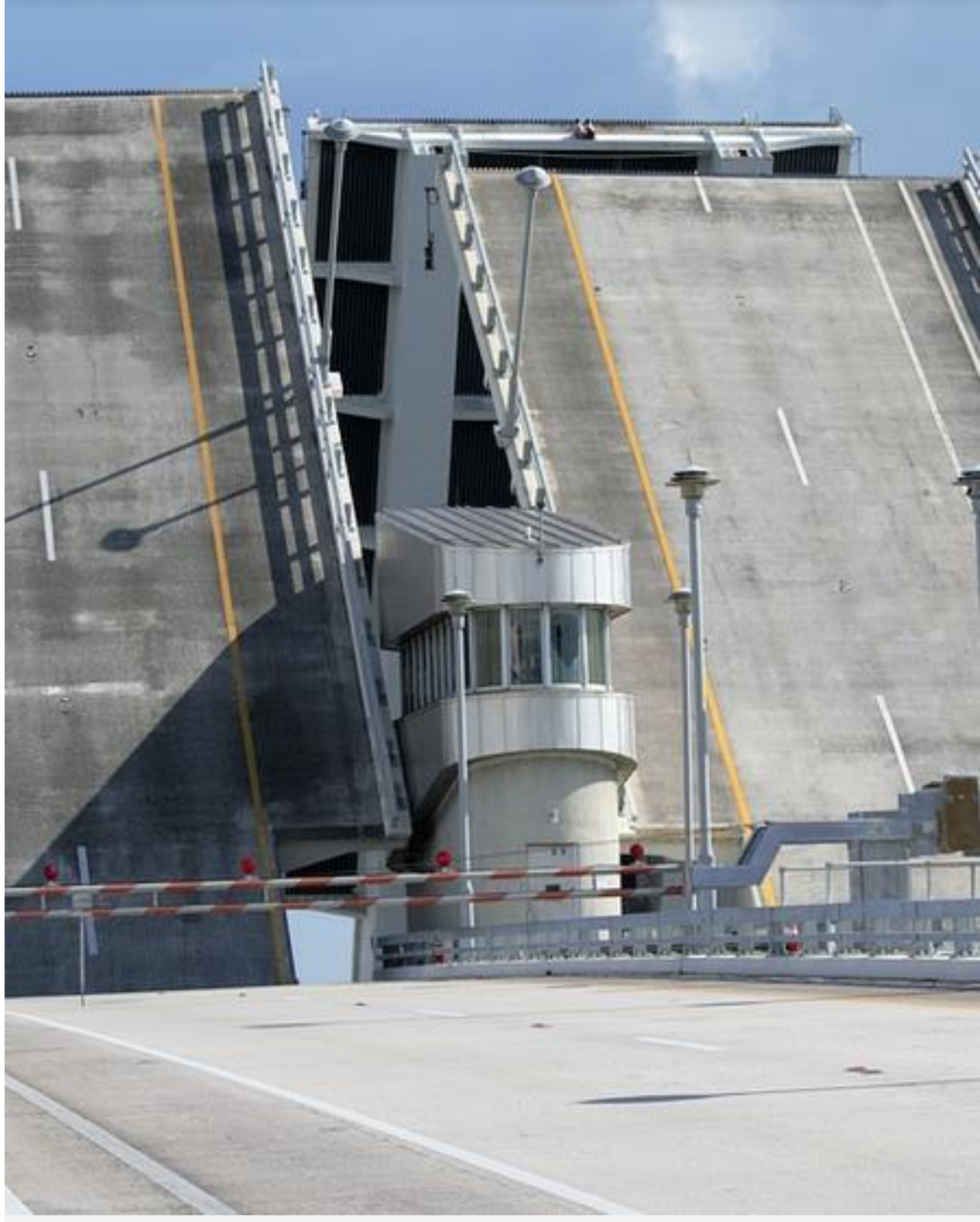
Often skills, roles and interest are in different teams…
we need to team up, collaborate or take over.



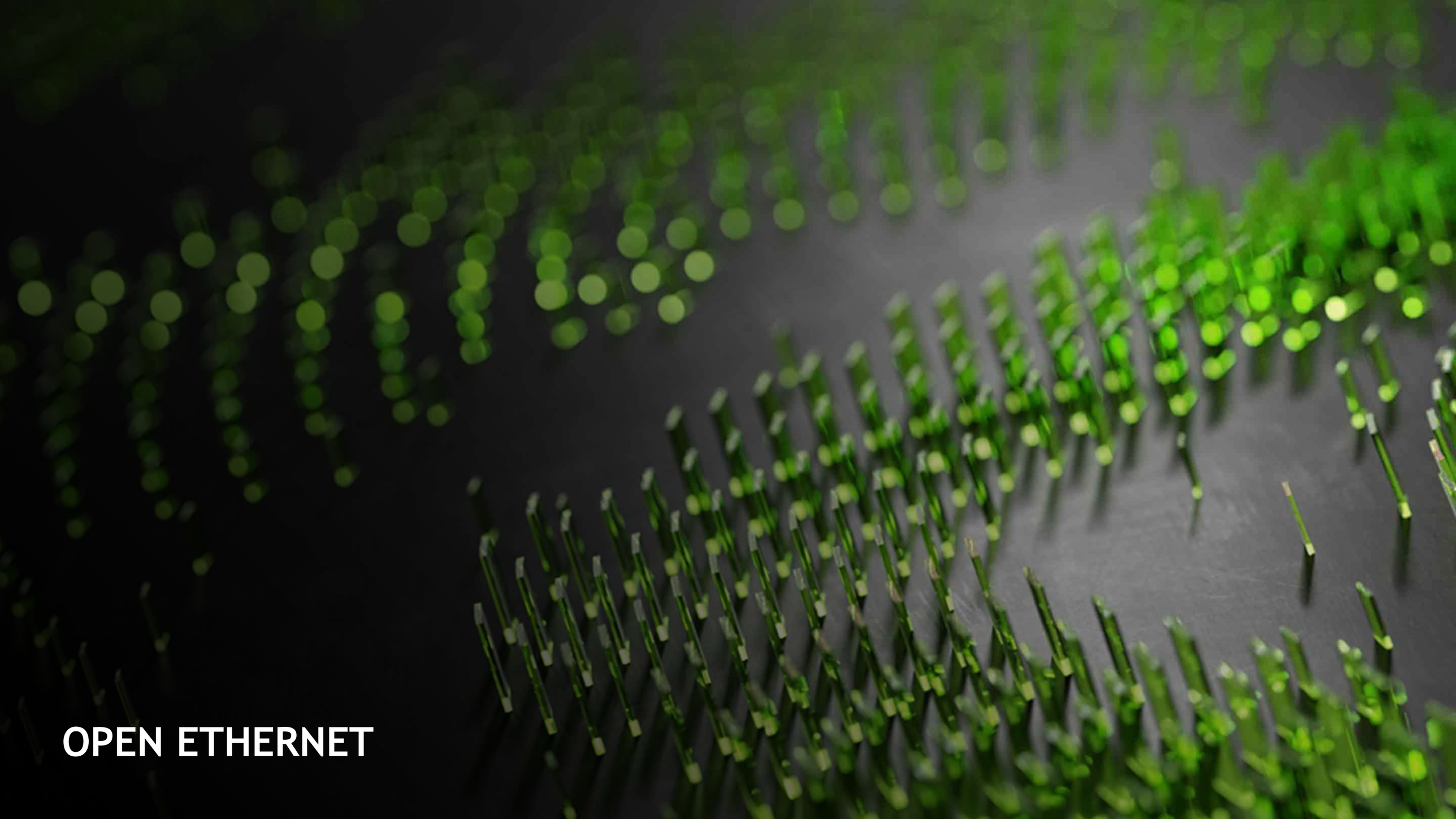
TWO TRACKS IN PARALLEL

Building Ansible Playbooks and Building Code-as-Infrastructure

Both tracks are mandatory for a working commute/ solution.



ALQ (20 min)


OPEN ETHERNET

OPEN ETHERNET INITIATIVE

Traditionally, Ethernet switches were a “locked” solution. Hardware and software were provided by the same vendor. The Open Ethernet initiative allows a complete separation of the switch hardware from the switch software.

By eliminating this dependency, Open Ethernet allows any software to run on any hardware:

- Freedom of choice
- Flexibility. Either hardware or software can be replaced or upgraded without changing the other components.
- Open-source solution

Traditional Ethernet Switch

Open Ethernet Switch

TL

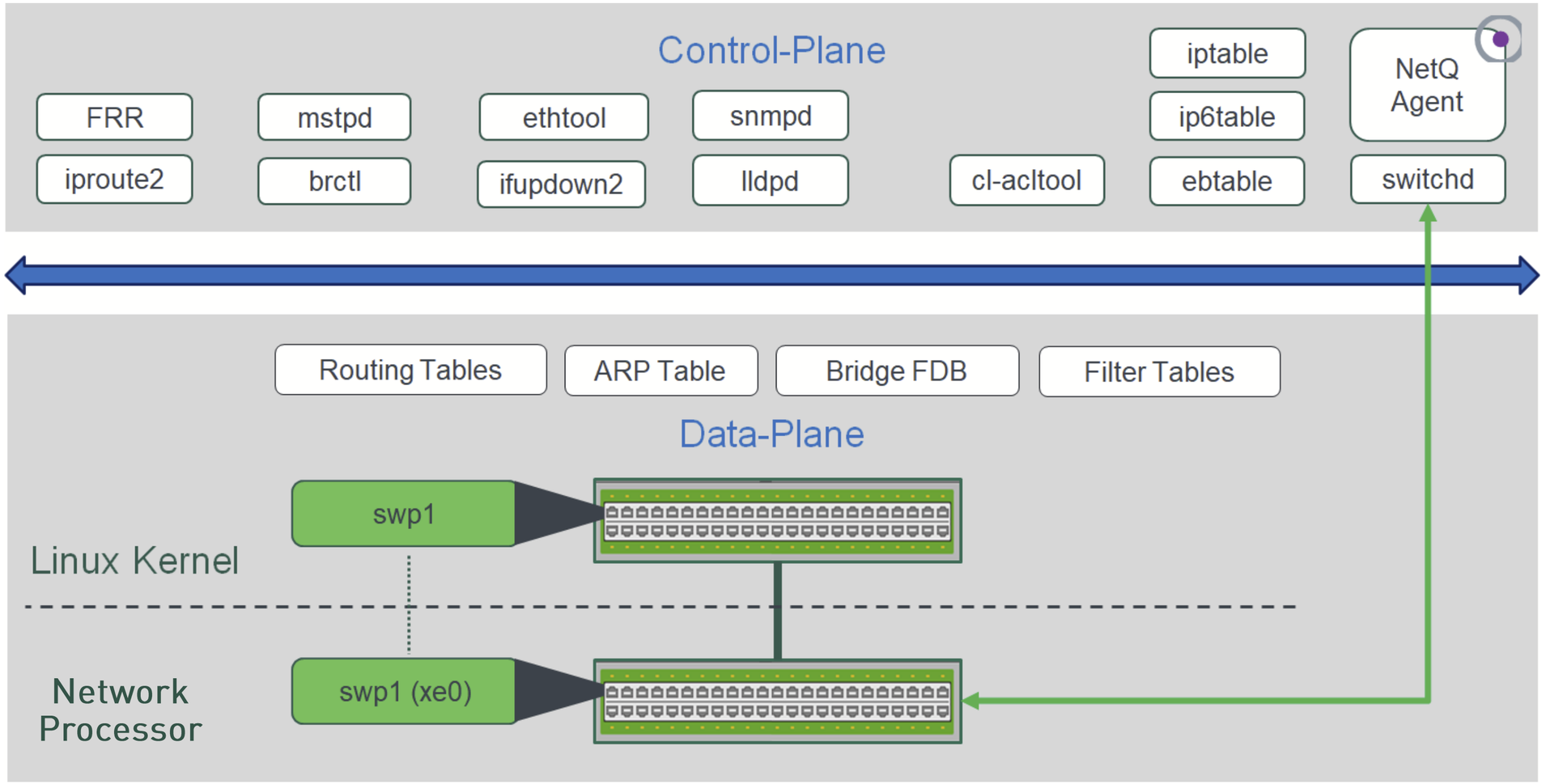
WHAT IS CUMULUS LINUX?

- An OS for Open Ethernet switches
- Native Linux distribution
- Open and proven, not proprietary, not “Linux-like” but "Linux"
- Based on Debian-Buster
- Makes a switch behave like a Linux server with many ports
- Provides Linux tools for switches:
- Server provisioning/management tools work on the switch

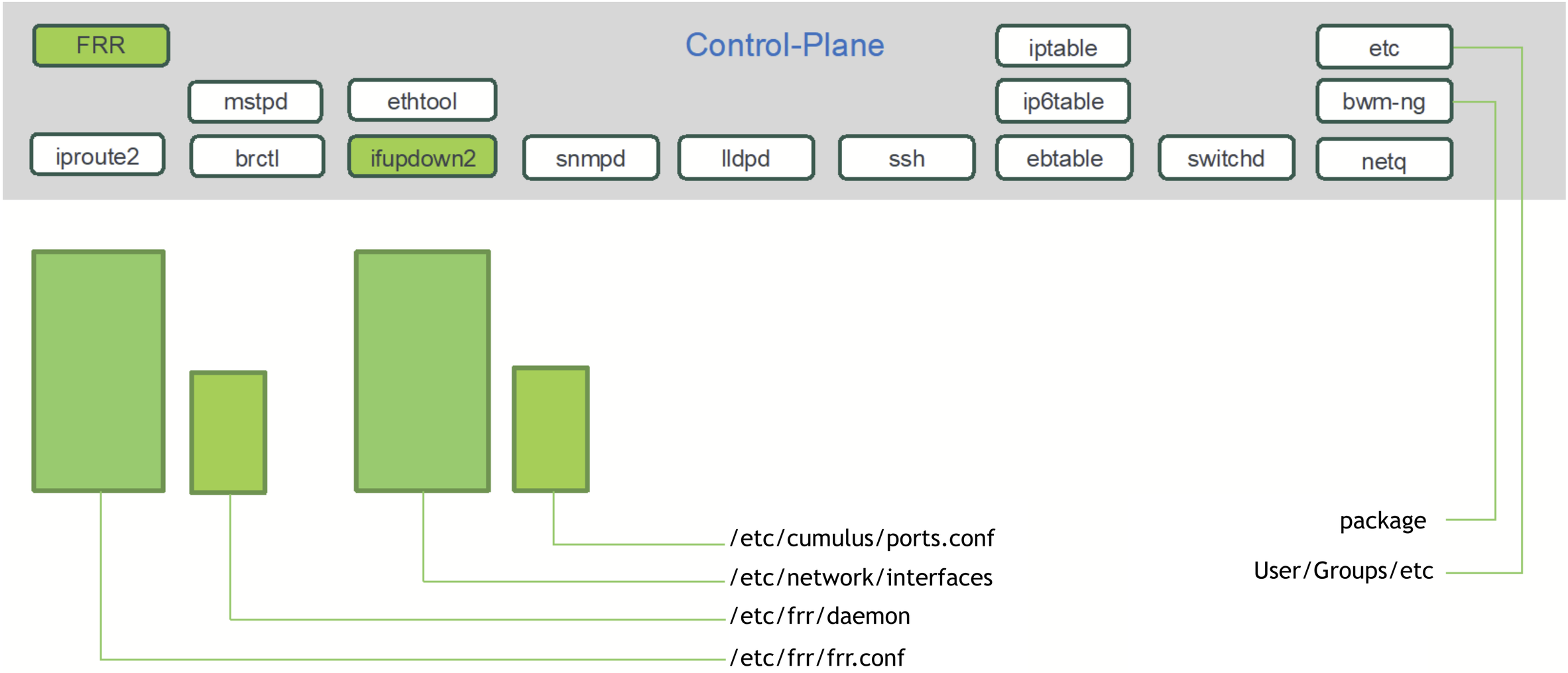
Cumulus Linux benefits:

- Linux admins can run the network
- Linux across the data center: server, storage & network

CUMULUS LINUX



CUMULUS LINUX



ALQ

NO ONE SIZE FITS ALL

You need to select the one road which works for you

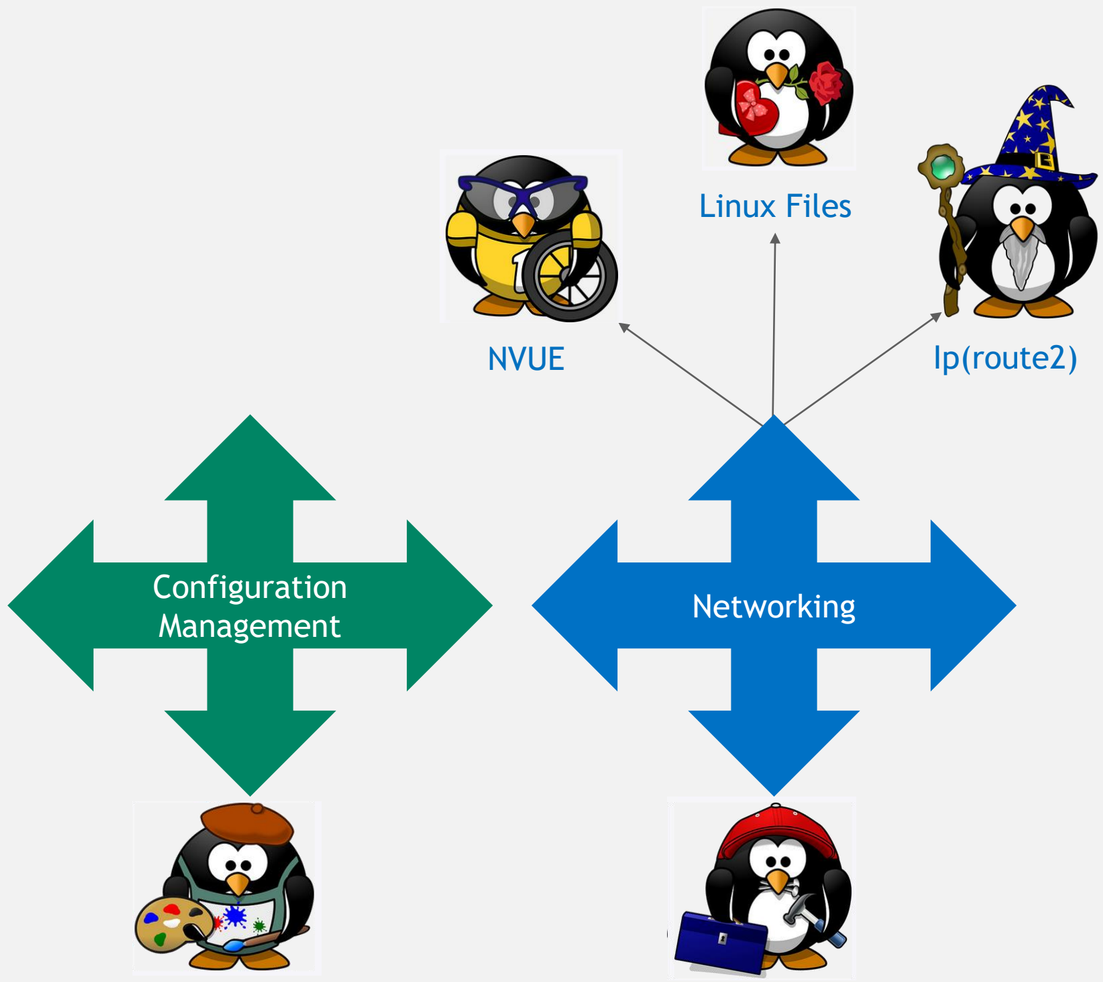
So, you started reading about options/sizes? Great!

Having asked for advice? Also great,
however, asking 3 often gives 4 different answers…

Yes, there are many roads leading to our destination:

- **NVUE (NVIDIA User Experience)**
- **Linux Files**
- **IP(ROUTE2)**
- **…**

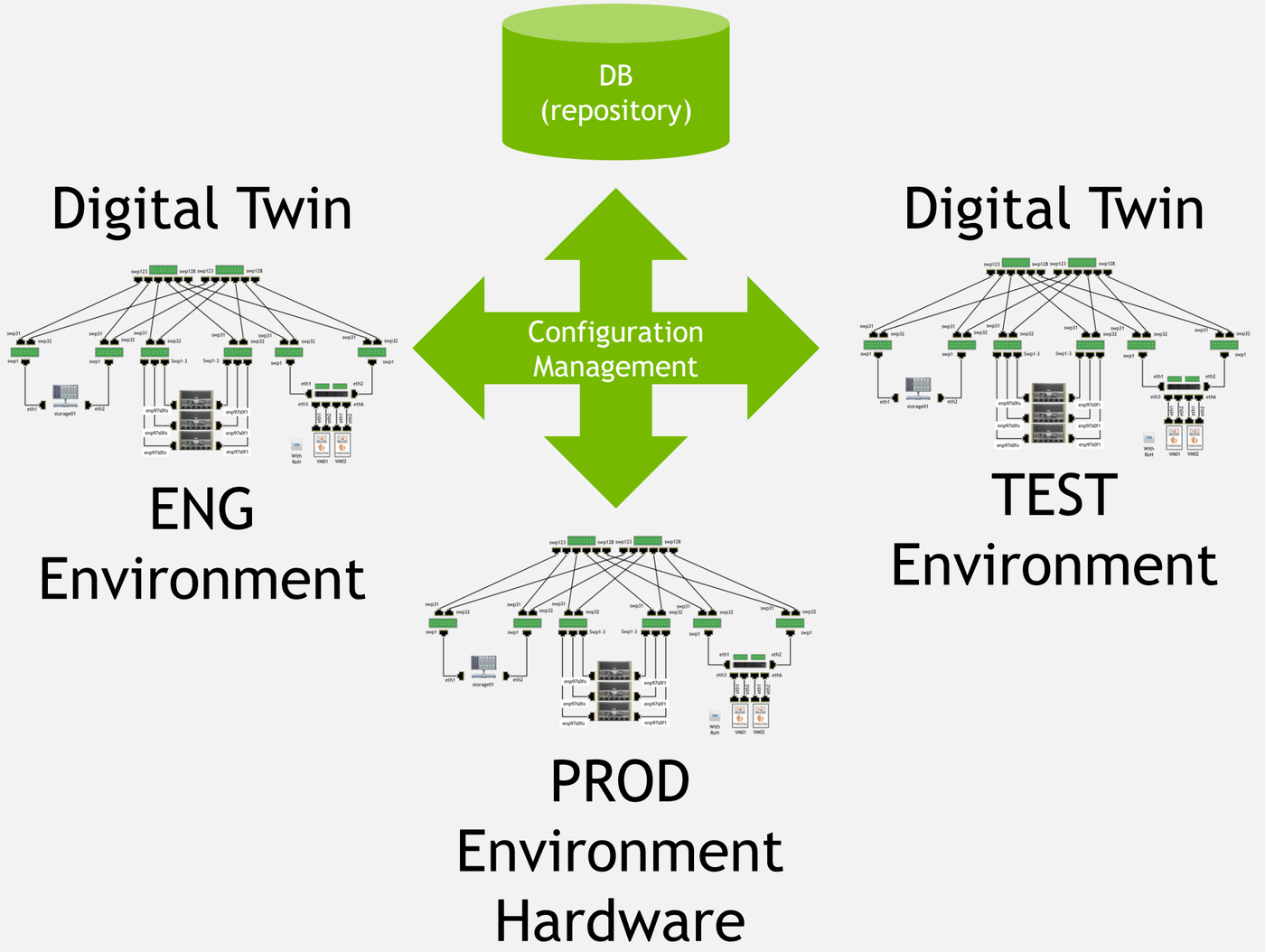
We will travel a bit on each of those roads, so you can enjoy and select.



ORCHESTRATION AND SVC

One “source of truth”

- One DB/Repository or source of truth
- Backup and Restore solved
- One Configuration Management
- Tool ==> one process for the entire DC



TL

SVC

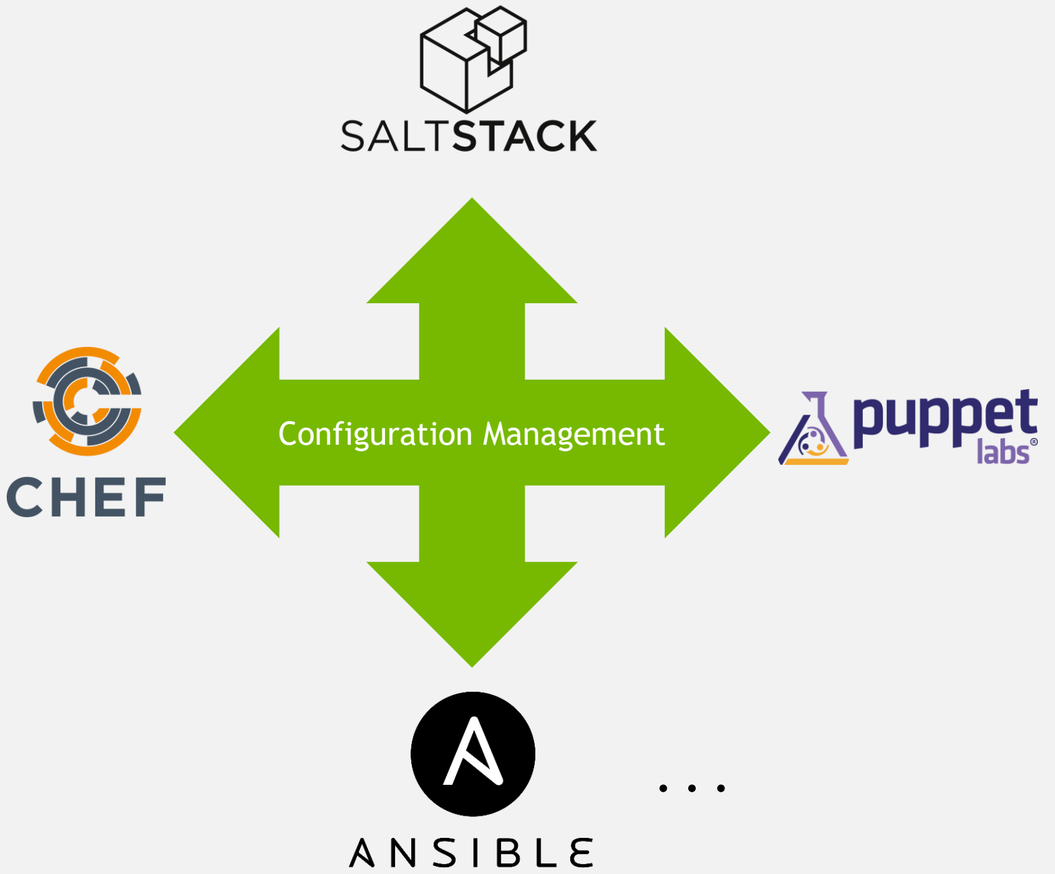
Source Version Control with git

ORCHESTRATION
Options

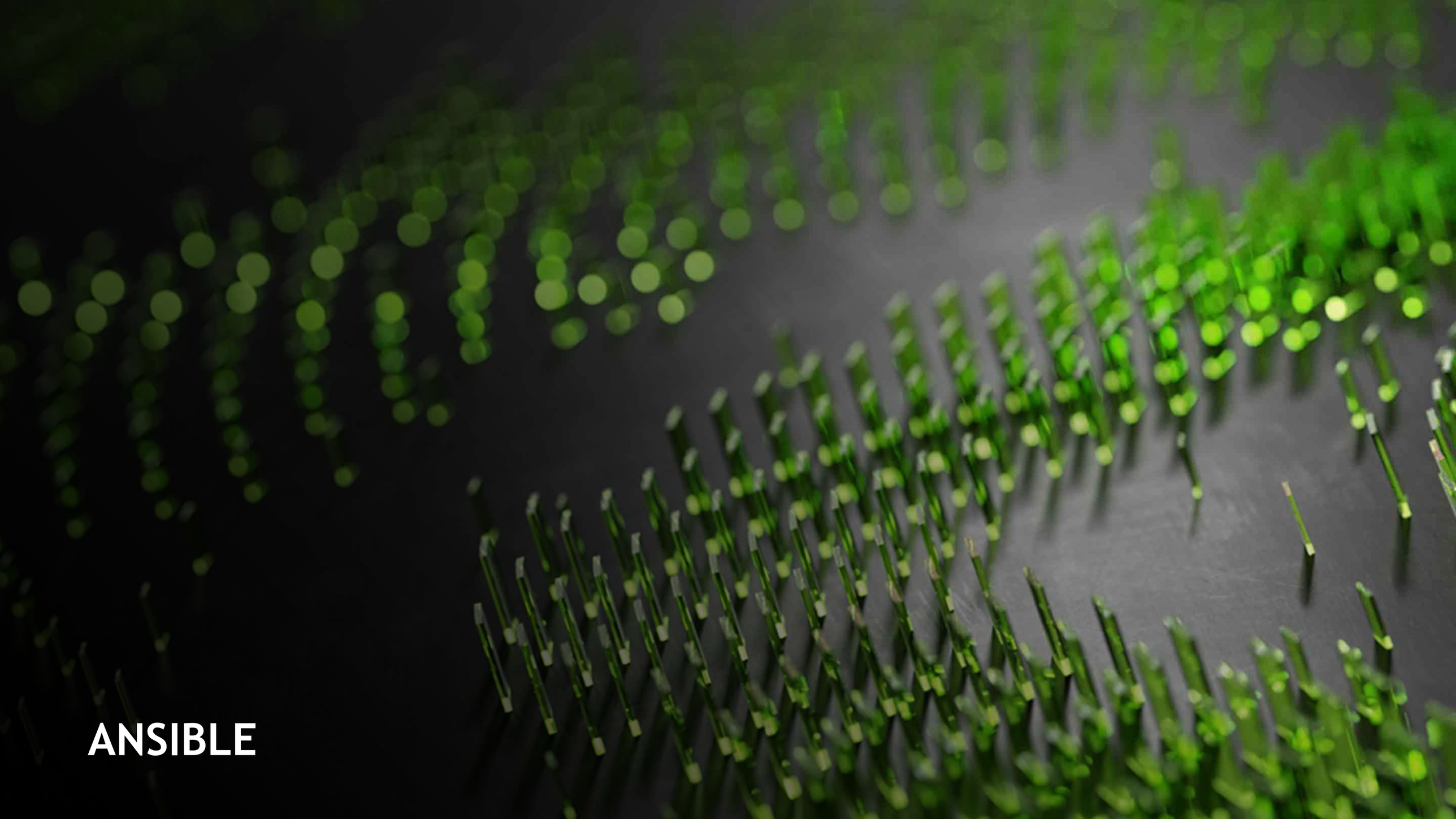
We will use Ansible during this week.

However, there are many other good/valid options like:

- **Chef**
- **SaltStack**
- **Puppet**
- **…**

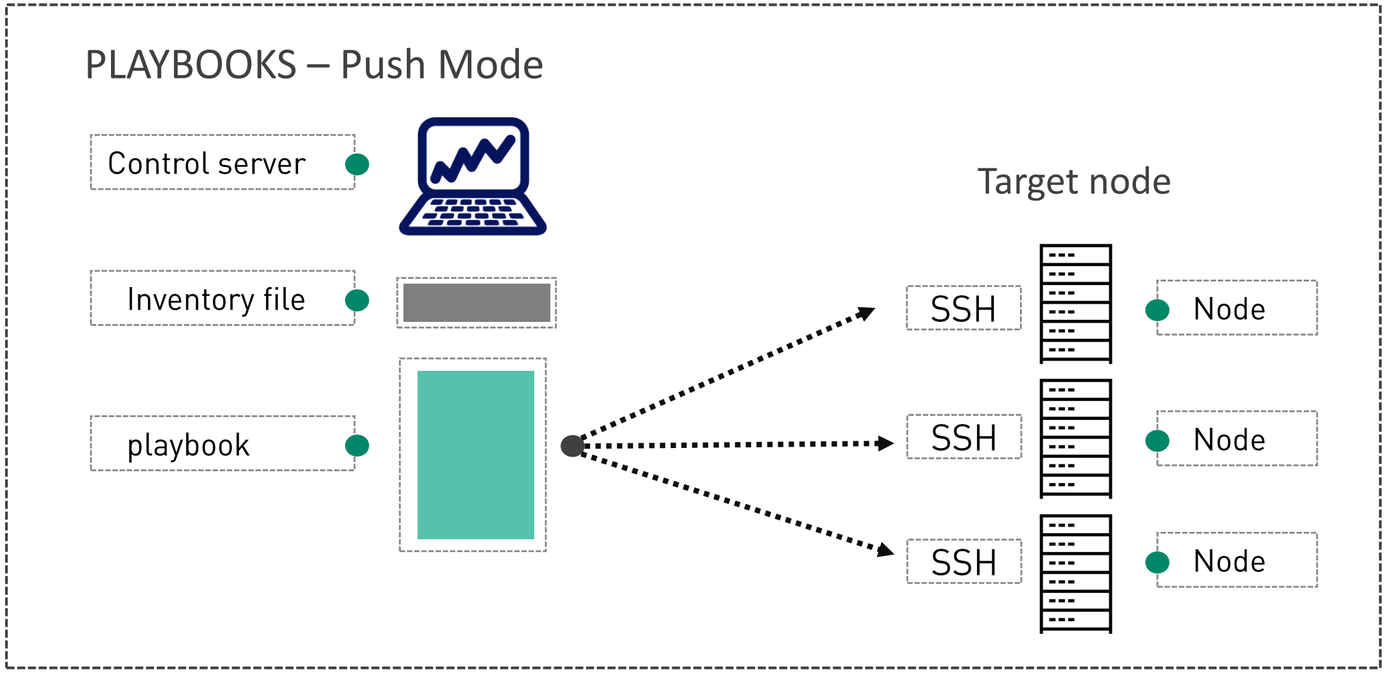


TL


ANSIBLE

ANSIBLE OVERVIEW

- Ansible is a configuration management and orchestration tool
- Ansible is installed on the control server only
- Ansible communicates with the clients and performs the required tasks using SSH
- Ansible can be used in two ways:
  - Ad-hoc commands - simple one-line commands
  - Playbooks – include multiple ordered tasks executed on the target nodes



TL

INSTALLING ANSIBLE

- Install Ansible on one (or more) server in the management network
- Two methods to install Ansible:
  1. Using Python repositories (PIP)

     Install from Python repository

     ```
     [root@mtlacad01 ~]# pip install ansible
     ```

  2. Using Linux repositories

     RedHat based distributions

     ```
     [root@mtlacad01 ~]# yum install ansible
     ```

     Debian based distributions

     ```
     [root@mtlacad01 ~]# apt-get install ansible
     ```


TL

ANSIBLE HOSTS FILE (inventory_hostname)

- Inventory file contains a list of all target nodes that are managed by Ansible.
- The File is located by default under :
  - `/etc/ansible/hosts`
- Hosts may be defined in the file as :
  - Discrete hosts definition
  - IP address or Hostname
  - Groups of hosts
  - Parent group may include multiple CHILDERN groups (even all fabric groups)

- Hosts or Hosts groups access parameters - USERS & PASSWORDS
  - We shell define the exact user & password of each group under the group : var section .

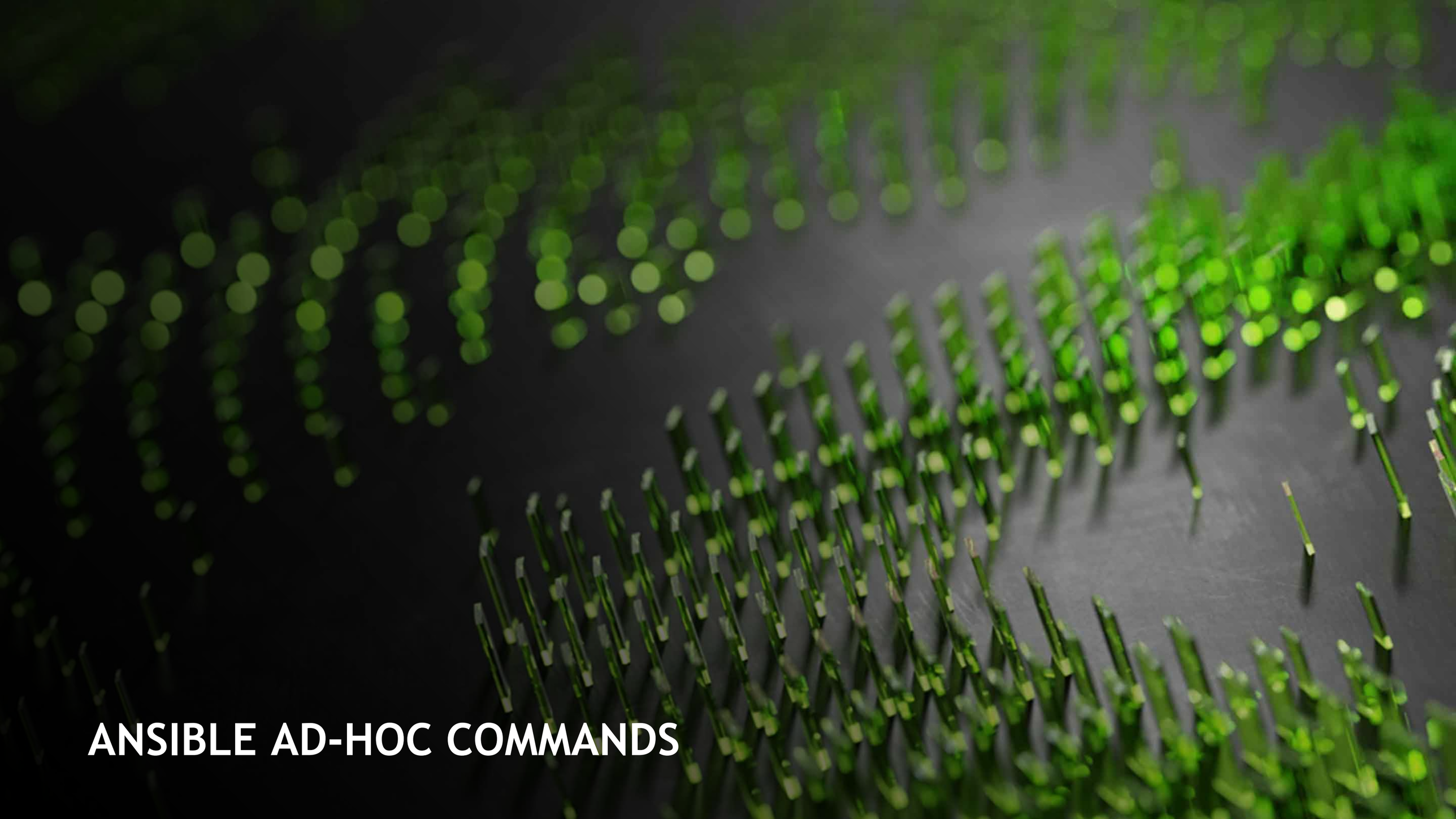
TL

ANSIBLE cfg FILE (configuration)

- Contains paths and references to ansible components.
- Certain settings in Ansible are adjustable via a configuration file. The stock configuration should be sufficient for most users, but there may be reasons you would want to change them.

- Changes can be made and used in a configuration file which will be processed in the following order:
  - ANSIBLE_CONFIG (an environment variable)
  - `ansible.cfg` (in the current directory)
  - `.ansible.cfg` (in the home directory)
  - `/etc/ansible/ansible.cfg`

TL



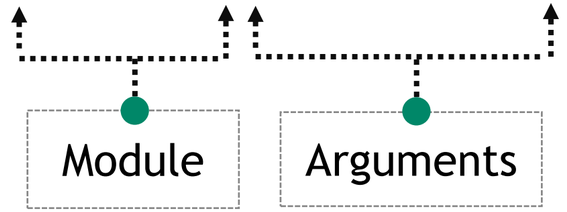
ANSIBLE AD-HOC COMMANDS

ANSIBLE Ad-hoc COMMANDS

- Ad-hoc commands are simple one-line commands executed on target nodes

  Run ad-hoc commands

```
[root@mtlacad01 ~]# ansible NODES-LIST -m MODULE -a ARG1 ARG2
```



- Modules take key=value arguments, space delimited
  - Some modules take no arguments
  - The **command** module's argument is a string representing the command to be executed

  Example:

```
ansible cl-spine3c -m ping
```


TL

‘copy & fetch’ MODULES EXAMPLE

Simple file copy (Backing up the hosts file)

```
[root@mtlacad01 ~]# ansible mtlacad03 -m copy -a "src=/etc/ansible/hosts dest=~/training/hosts”
```

Destination path on the remote host

Use ‘copy’ module

Source file on the local host

Change Message of The Day

```
[root@mtlacad01 ~]# ansible target_node -m copy -a 'dest=/etc/motd content="This server is managed by Ansible"'
```

Use ‘fetch’ module

Copy content (text) to destination file

Fetch the switch ports configuration

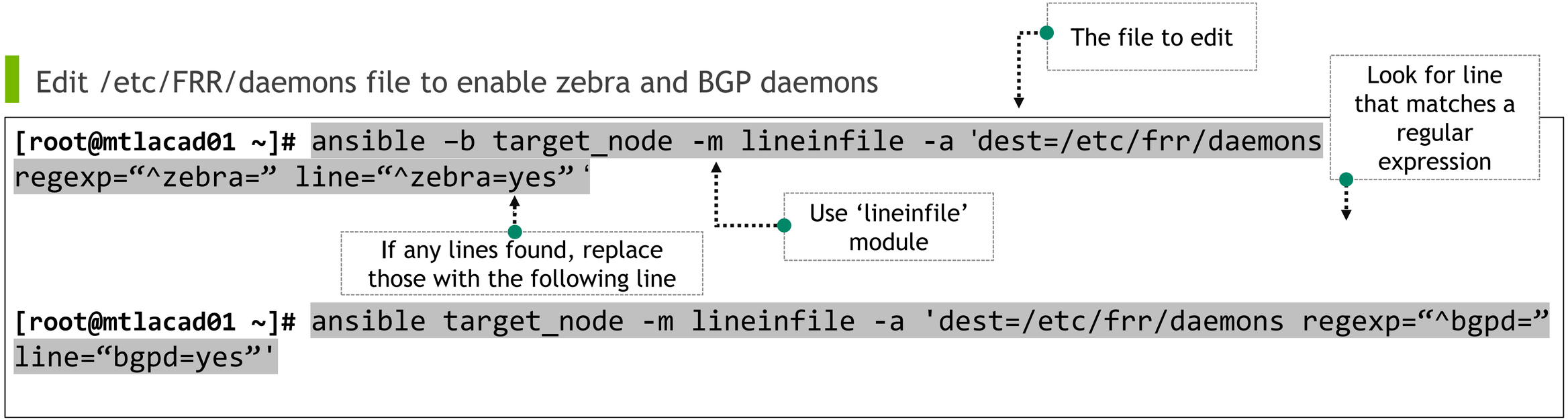
```
[root@mtlacad01 ~]# ansible cl-spine3a -m fetch -a ‘src=/etc/cumulus/ports.conf
dest=/~/training/port_conf_backup.conf flat=yes’
```

Only fetch files (instead of entire directory)

TL

‘lineinfile’ MODULE EXAMPLE

Edit /etc/FRR/daemons file to enable zebra and BGP daemons



- If no lines that matches the regex are found, the line will be added as a new line in the destination file.
- After the files were edited, the FRR service needs to be restarted

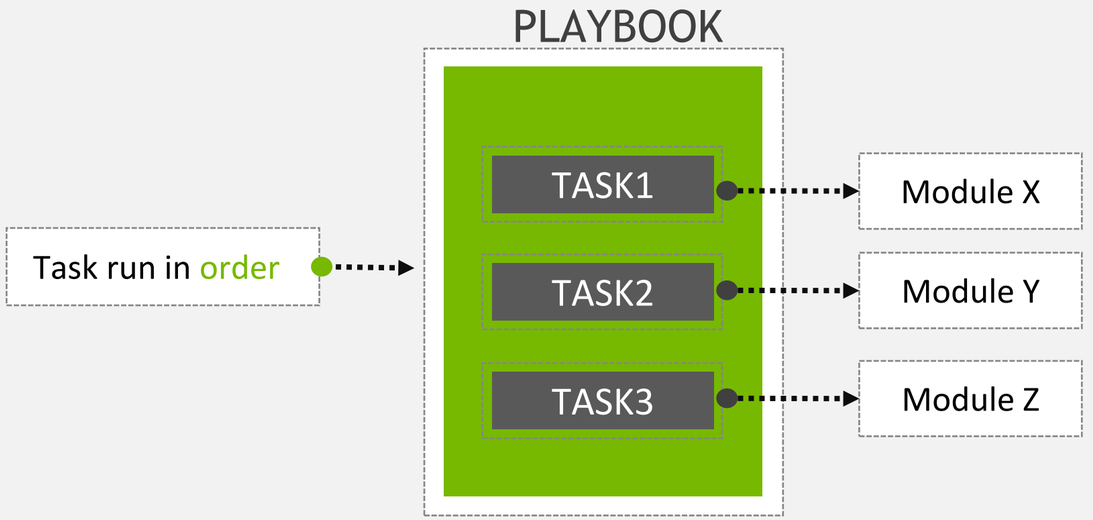
```
ansible -b  cl-leaf1b -m lineinfile -a 'dest=/etc/frr/daemons regexp=“^zebra=” line=“^zebra=yes”'
```


TL


ANSIBLE PLAYBOOKS

ANSIBLE PLAYBOOKS

- An Ansible playbook contains one or multiple plays, each of which define **the work** to be done for configuration on a target node
- Ansible plays use modules that execute tasks on the target nodes
- Ansible plays are written in YAML.
  - YAML (Yet Another Markup Language) is a human readable data format language (like JSON or XML)
  - YAML is commonly used for configuration files



TL

YAML SYNTAX

- YAML files can begin with three dashes (**`---`**) indicating the start of the document and end with three dots **`(...)`** indicating the end of a document
- Comments begin with the pound sign (**`#`**), can start anywhere on a line and continue until the end of the line
- For Ansible, nearly every YAML file starts with a list. All members of a list are lines beginning at the same indentation level starting with a dash and a space (**`-`**)
- Each item in the list is called “dictionary”. A dictionary is represented in a simple **`key: value`** form (the colon must be followed by a space)

YAML file syntax

```
---
# Employee records
- maya:
    name: Maya Mintz
    job: Manager
    skills:
      - managing
      - Python
- yair:
    name: Yair Iluz
    job: Developer
    skills:
      - ansible
      - video_editing
...
```


TL

ANSIBLE PLAYBOOK EXAMPLE

NVUE command example

Ansible playbook file

```
---
- hosts: leaf01
  name: create bridge, set loopback and enable switch ports
  become: yes
  gather_facts: no
  tasks:
    - name: nvue set items
      shell: nv set {{ item }}
      with_items:
      - interface lo ip address 192.168.0.1/32
      - interface swp1,31-32
      - bridge domain br_A vlan 10,11
      - interface swp1 bridge domain br_A access 10
      - platform hostname value leaf01

    - name: activate staging buffer
      shell: nv config apply -y
```

NVUE YAML (startup.yaml)

```
- set:
    bridge:
      domain:
        br_A:
          vlan:
            '10': {}
            '11': {}
    platform:
      hostname:
        value: leaf01
    interface:
      lo:
        ip:
          address:
            192.168.0.1/32: {}
        type: loopback
      swp1:
        type: swp
        bridge:
          domain:
            br_A:
              access: 10
      swp31:
        type: swp
      swp32:
        type: swp
```

ALQ

ANSIBLE PLAYBOOK EXAMPLE

Templating by using Jinja, two examples

Optional step-07 covering
roles, templates and vars
"for the advanced students"

Complex example (Cumulus consulting team)

```
{% if comments|default(False) %}
###############
# Loopback
###############
{{ loopback | default() | to_nice_yaml | comment(prefix='# lopback:', postfix='') -}}
{% endif %}
auto lo
iface lo inet loopback
{% for loopback in loopback.ips|default() %}
    address {{ loopback }}
{% else %}
    address 127.0.0.1
{% endfor %}
{% if loopback.clag_vxlan_anycast_ip is defined %}
    clagd-vxlan-anycast-ip {{ loopback.clag_vxlan_anycast_ip }}
{% endif %}
{% if loopback.vxlan_local_tunnel_ip is defined %}
    vxlan-local-tunnelip {{ loopback.vxlan_local_tunnel_ip }}
{% endif %}
{% for extra in loopback.extras|default() %}
    {{ extra }}
{% endfor %}

{##}
```

https://gitlab.com/cumulus-consulting/goldenturtle/cumulus_ansible_modules/-/blob/master/roles/interfaces/templates/features/loopback.j2

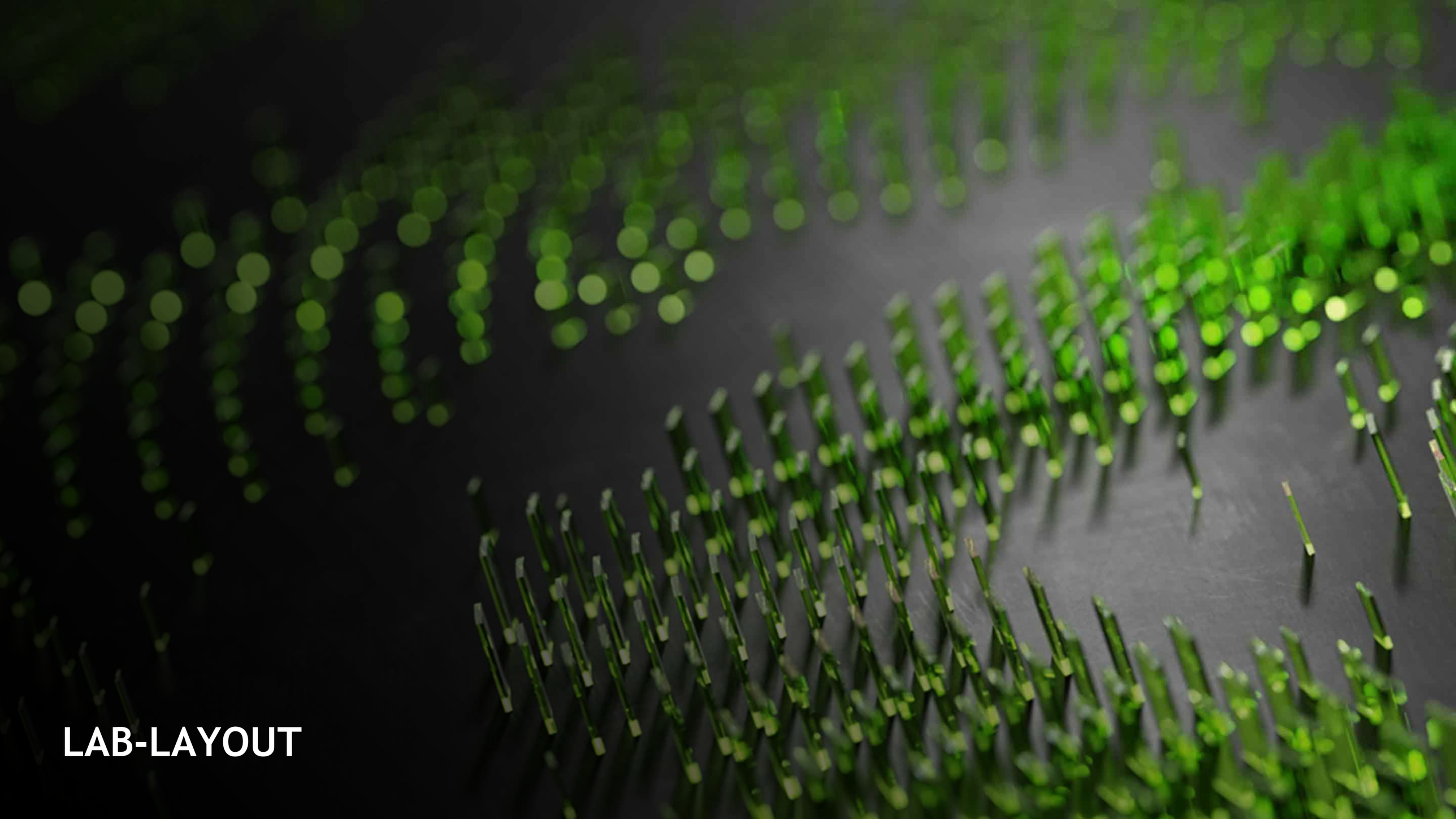
Simple template (leaf01-if.j2) setting a variable and using it

```
{% set ip_lo = "7.7.7.7/32" %}

auto lo
iface lo inet loopback
    address {{ ip_lo }}

```

Standard Ansible modules like copy and template

```
tasks:
    - name: render file via template
      template:
        src: /home/cumulus/ON-15/step-06c/templates/leaf01-if.j2
        dest: /etc/network/interfaces
```

ALQ

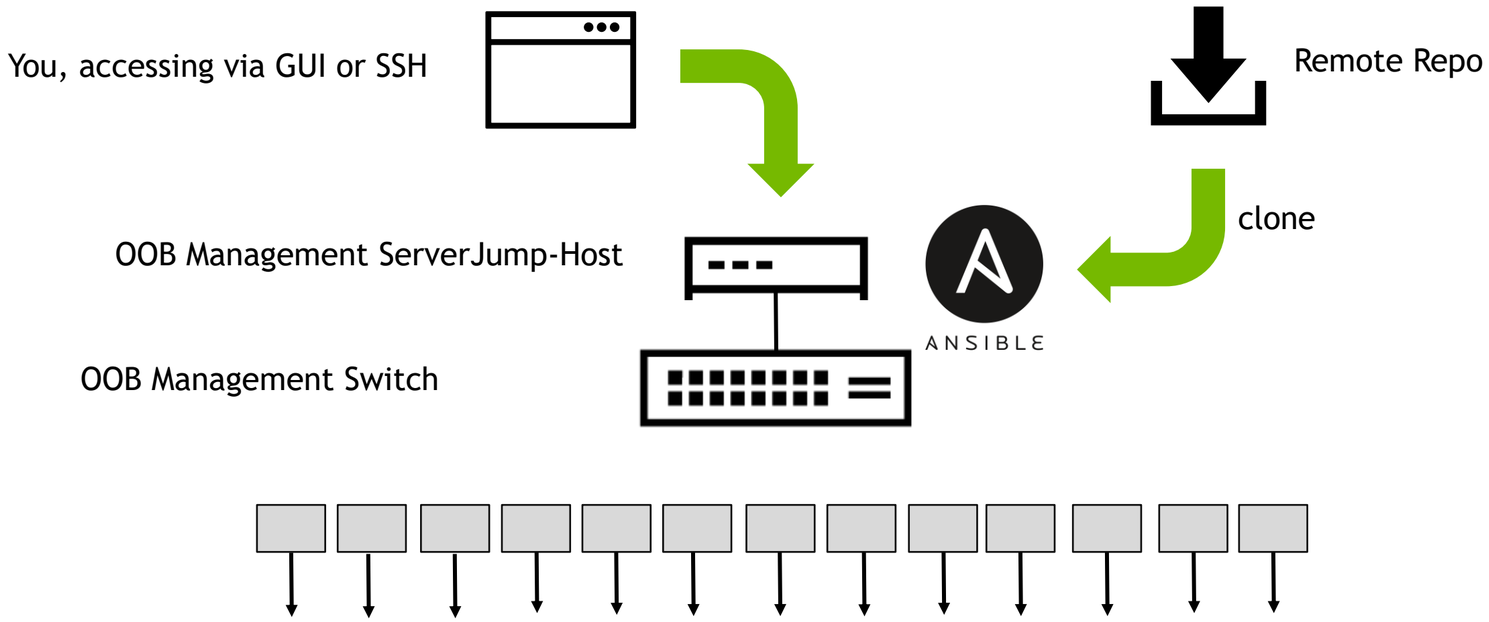

LAB-LAYOUT

DIGITAL TWIN

Lab-Architecture GTC-2021
Access

Step1: registration via
www.nvidia.com must be in place

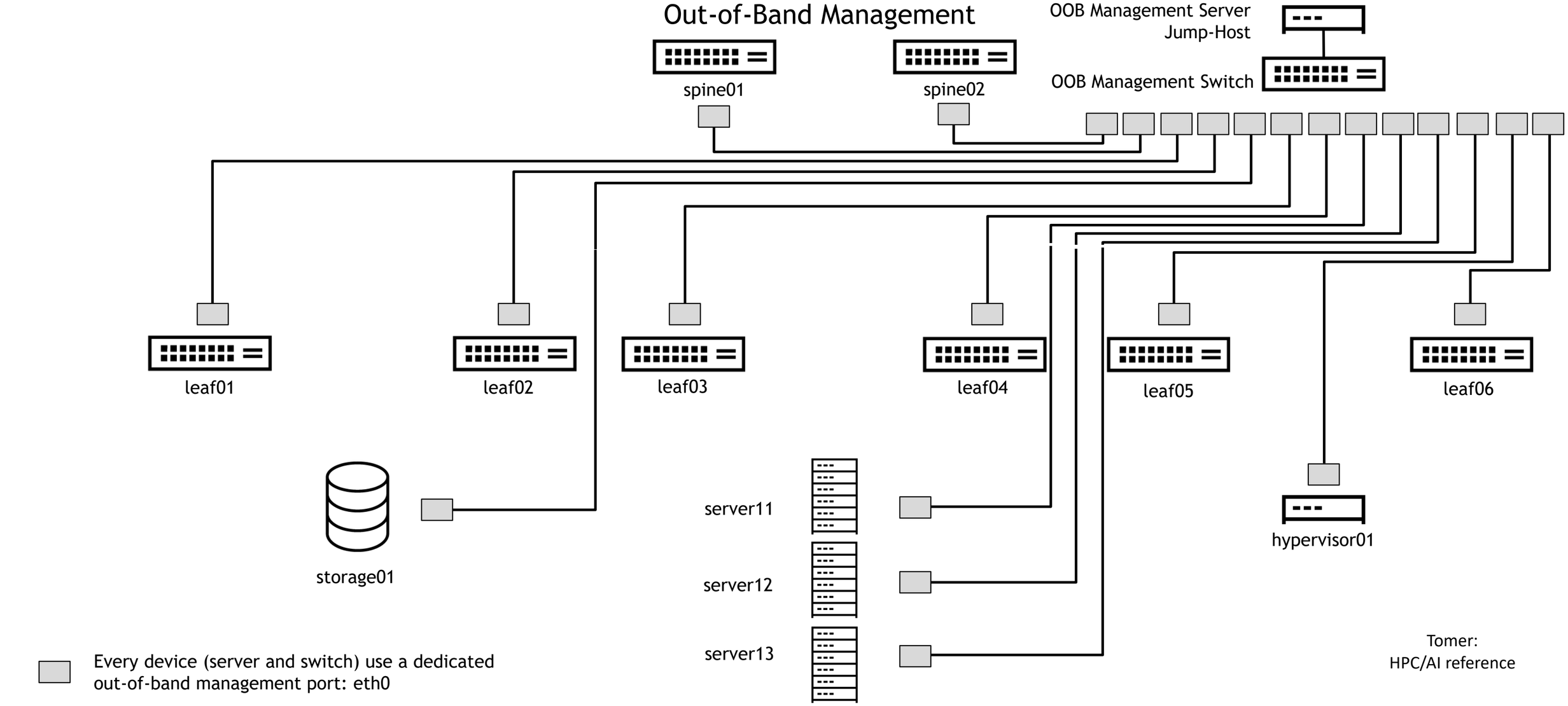
www.air.nvidida.com
$ ssh –p …
cumulus@worker....air.nvidia.com



ALQ

DIGITAL TWIN

Lab-Architecture GTC-2021
Out-of-Band Management



Every device (server and switch) use a dedicated out-of-band management port: eth0

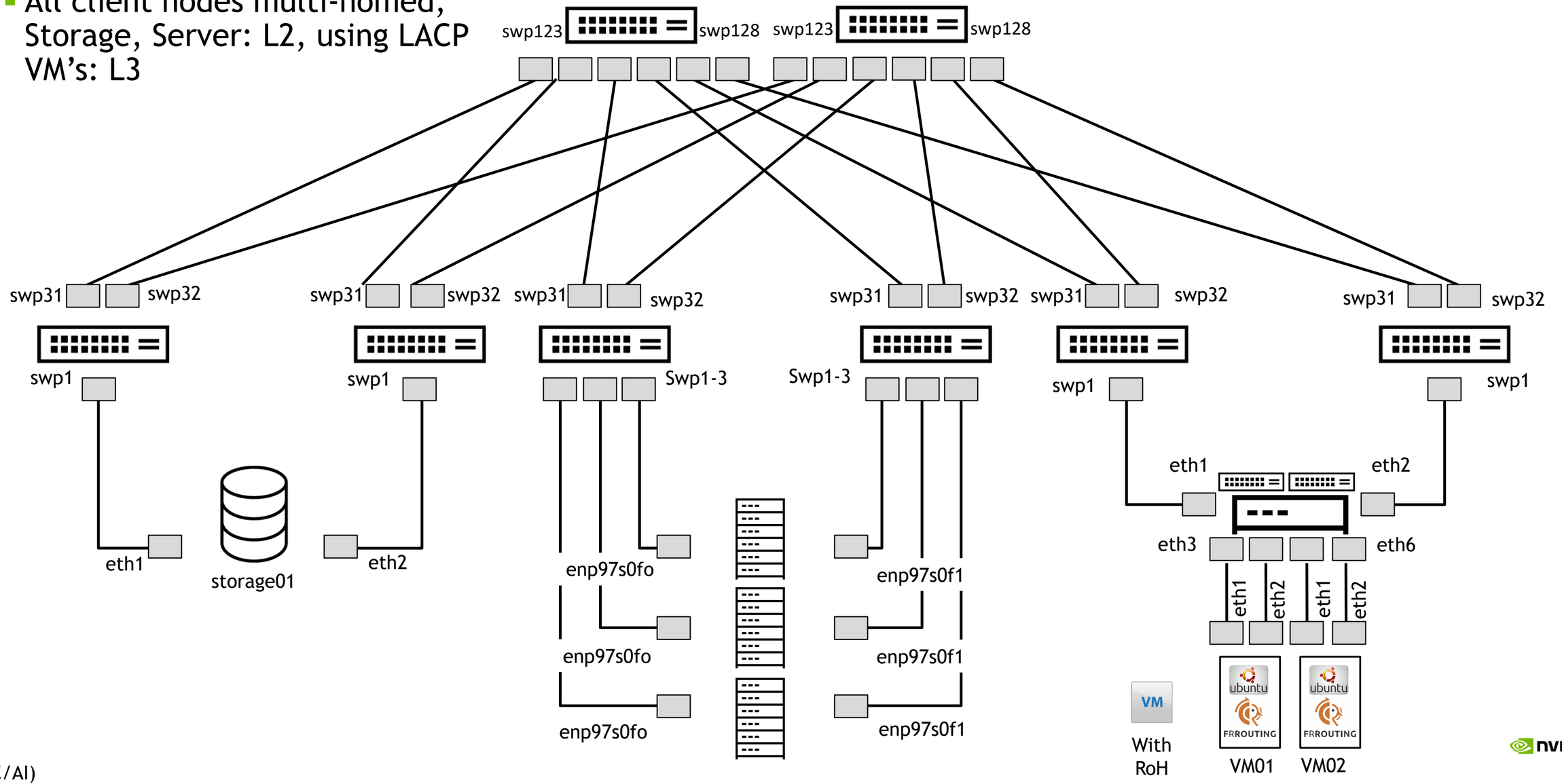
Tomer:
HPC/AI reference

ALQ

DIGITAL TWIN

Lab-Architecture GTC-2021

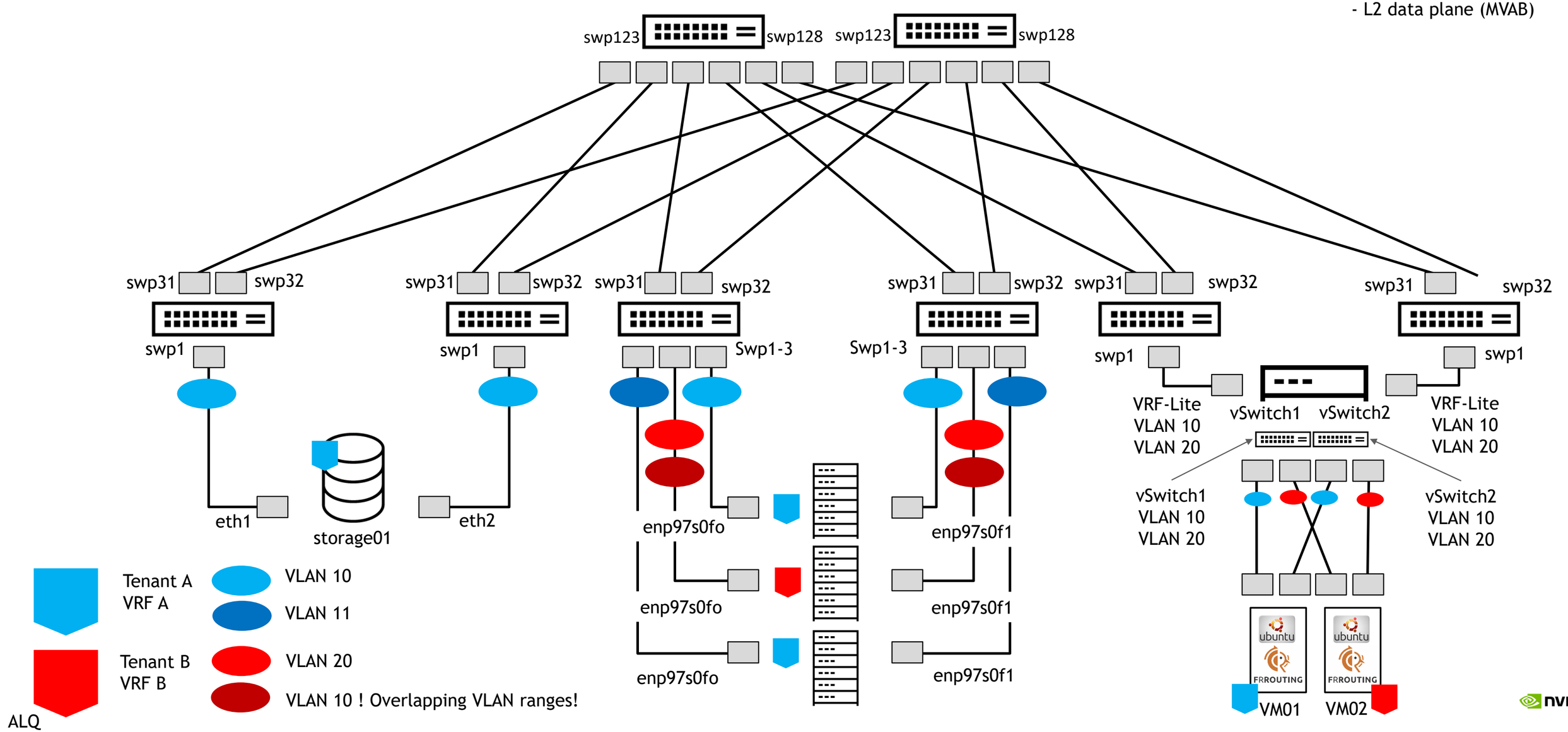
- All client nodes multi-homed, Storage, Server: L2, using LACP
  VM's: L3



DIGITAL TWIN

Lab-Architecture GTC-2021
Two Tenants

Note:
Multi-Tenant "domains"
- L3 control plane (BGP)
- L3 data plane (VRF)
- L2 data plane (MVAB)



DIGITAL TWIN

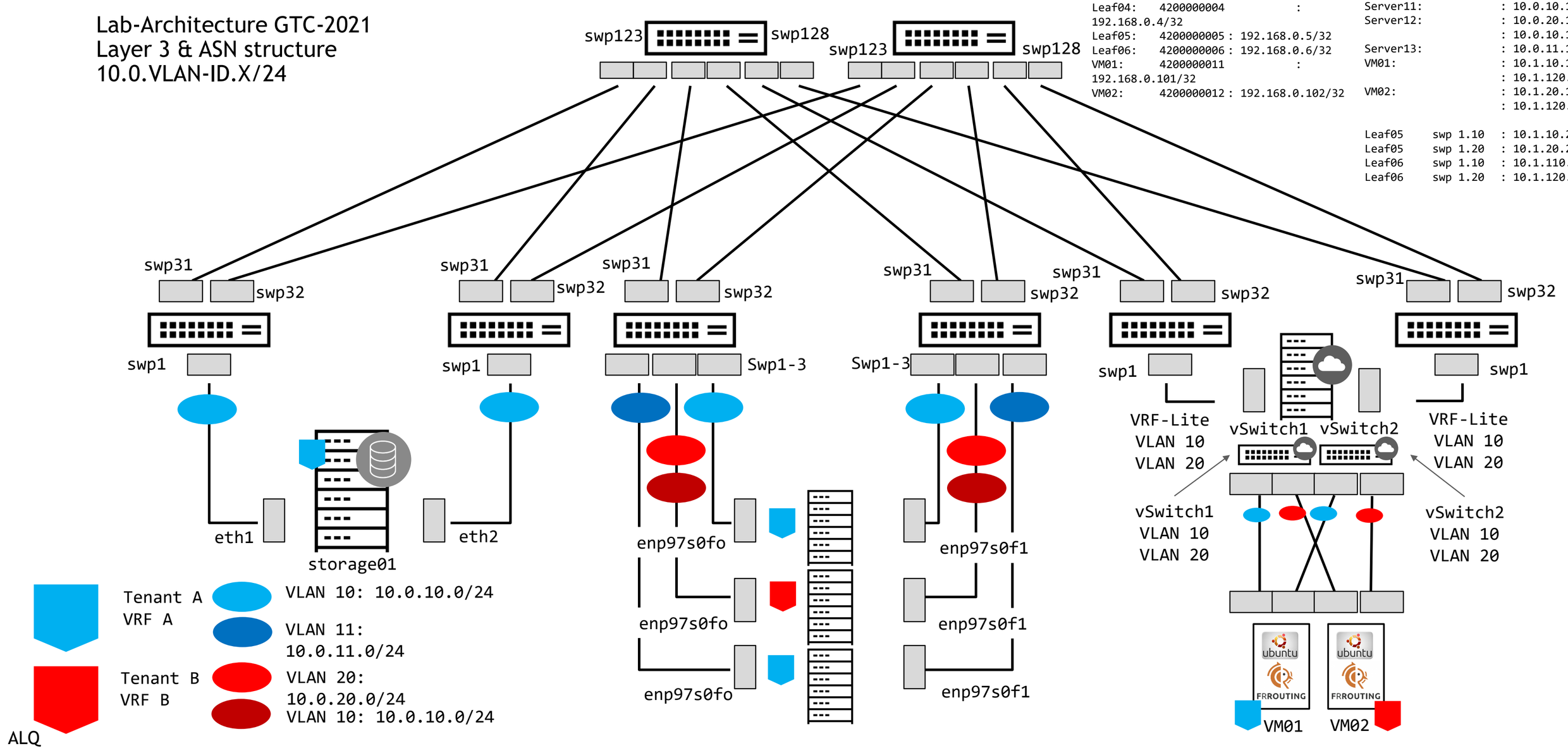
Lab-Architecture GTC-2021
Layer 3 & ASN structure
10.0.VLAN-ID.X/24

ANS & Loopbacks:

```
Spine01:   4200000201 : 192.168.0.201/32
Spine02:   4200000202           :
192.168.0.202/32
Leaf01:    4200000001           :
192.168.0.1/32
Leaf02:    4200000002 : 192.168.0.2/32
Leaf03:    4200000003           :
192.168.0.3/32
Leaf04:    4200000004           :
192.168.0.4/32
Leaf05:    4200000005 : 192.168.0.5/32
Leaf06:    4200000006 : 192.168.0.6/32
VM01:      4200000011           :
192.168.0.101/32
VM02:      4200000012 : 192.168.0.102/32
```

L3 Structure:

```
Anycast Addresses     : a.b.c.254/24
SVI                   : a.b.c.240-253/24
Reserved:             : a.b.c.1-9/24
Clients:              : a.b.c.10-199/24
Reserve:              : a.b.c.200-239/24
Multicast (BUM):      : 239.0.0.VLAN-ID

Storage01:            : 10.0.10.1/24
Server11:             : 10.0.10.11/24
Server12:             : 10.0.20.12/24
                      : 10.0.10.12/24
Server13:             : 10.0.11.13/24
VM01:                 : 10.1.10.101/24
                      : 10.1.120.101/24
VM02:                 : 10.1.20.102/24
                      : 10.1.120.102/24

Leaf05    swp 1.10    : 10.1.10.205/24
Leaf05    swp 1.20    : 10.1.20.205/24
Leaf06    swp 1.10    : 10.1.110.206/24
Leaf06    swp 1.20    : 10.1.120.206/24
```



DIGITAL TWIN

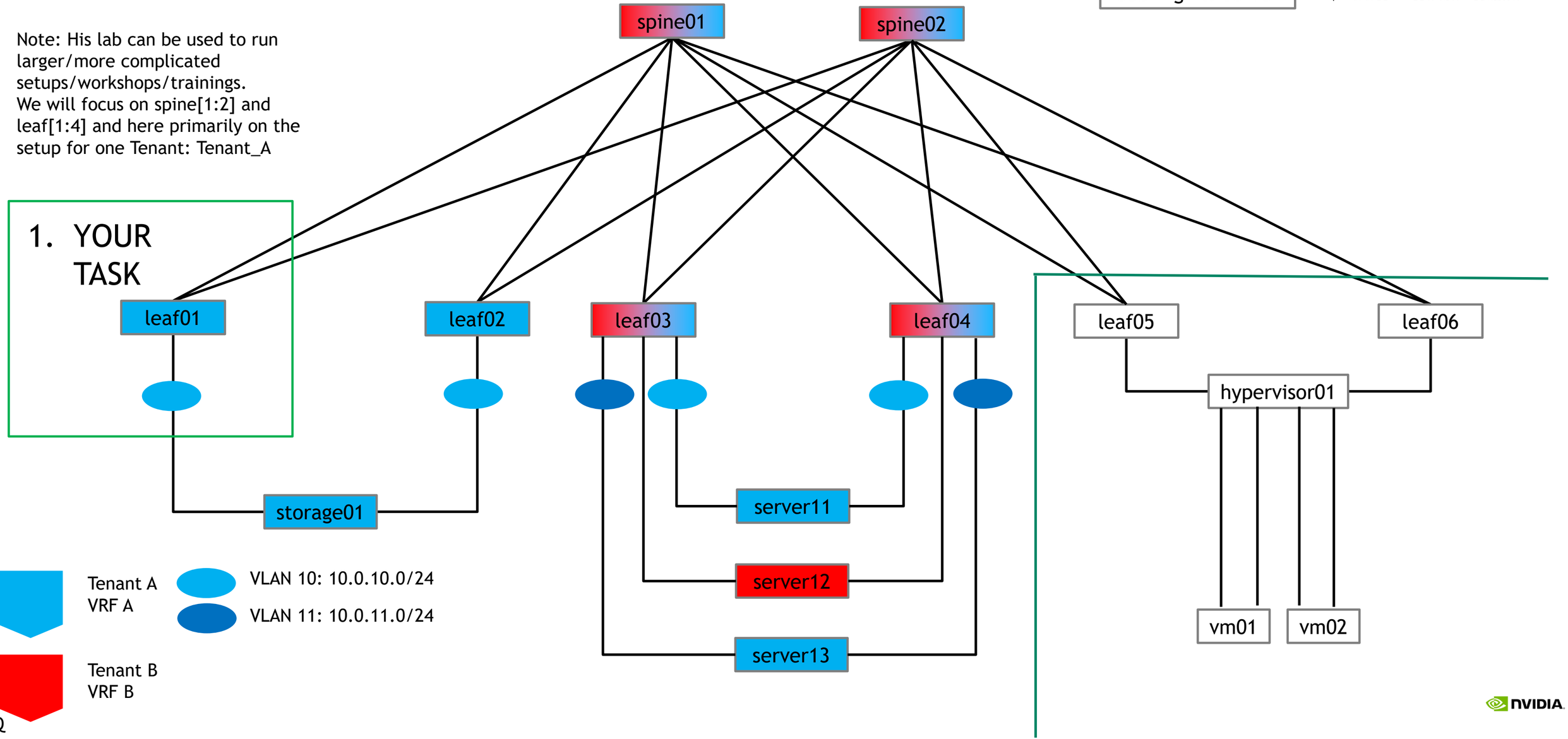
OUR FOCUS TODAY

Lab-Architecture GTC-2021
Device Selection or focus

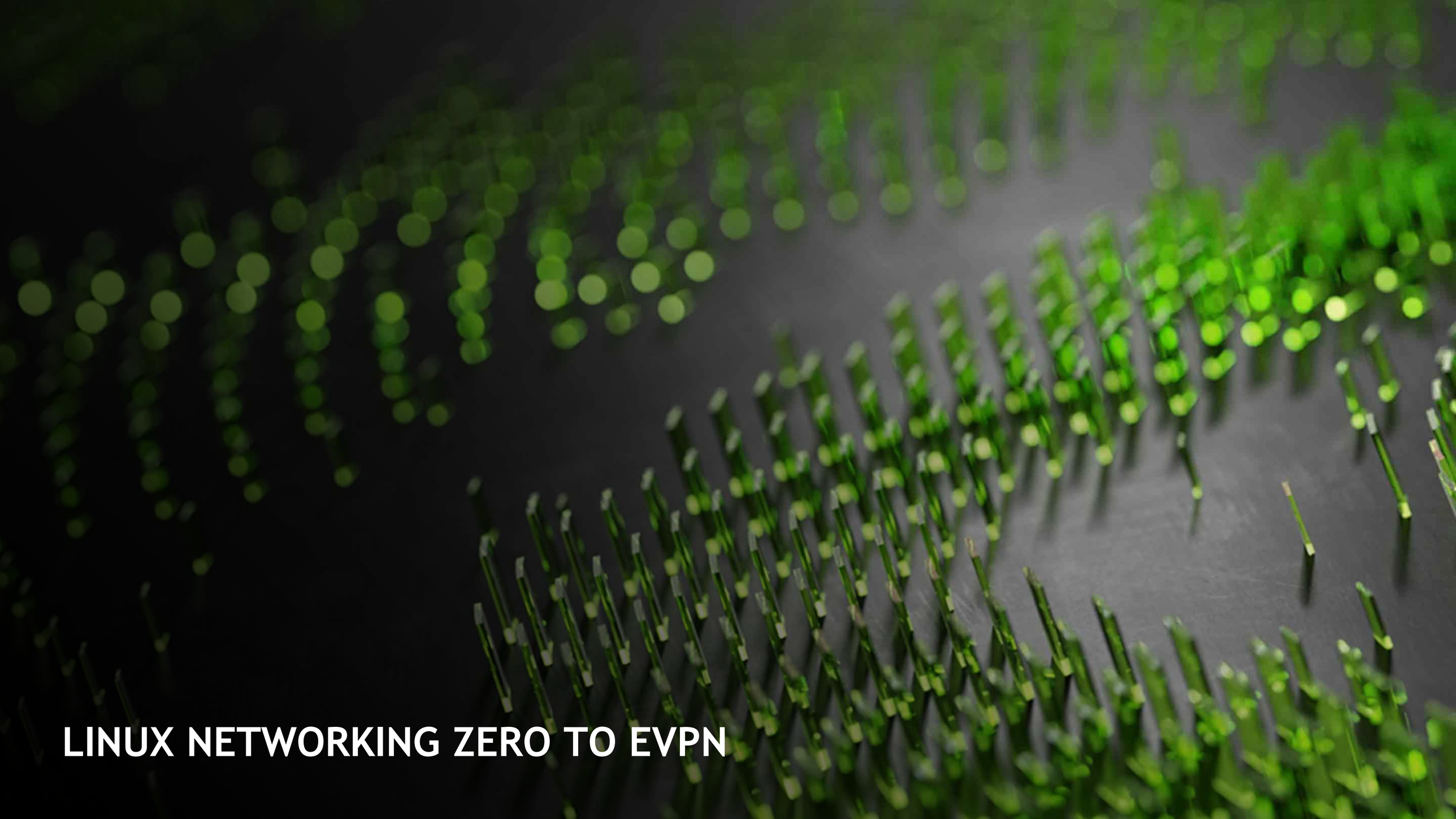
oob-mgmt.-server

```
$ ssh <node-name>
```

Note: His lab can be used to run larger/more complicated setups/workshops/trainings.
We will focus on spine[1:2] and leaf[1:4] and here primarily on the setup for one Tenant: Tenant_A



ALQ



LINUX NETWORKING ZERO TO EVPN

ZERO-TO-EVPN

Easy Linux Networking

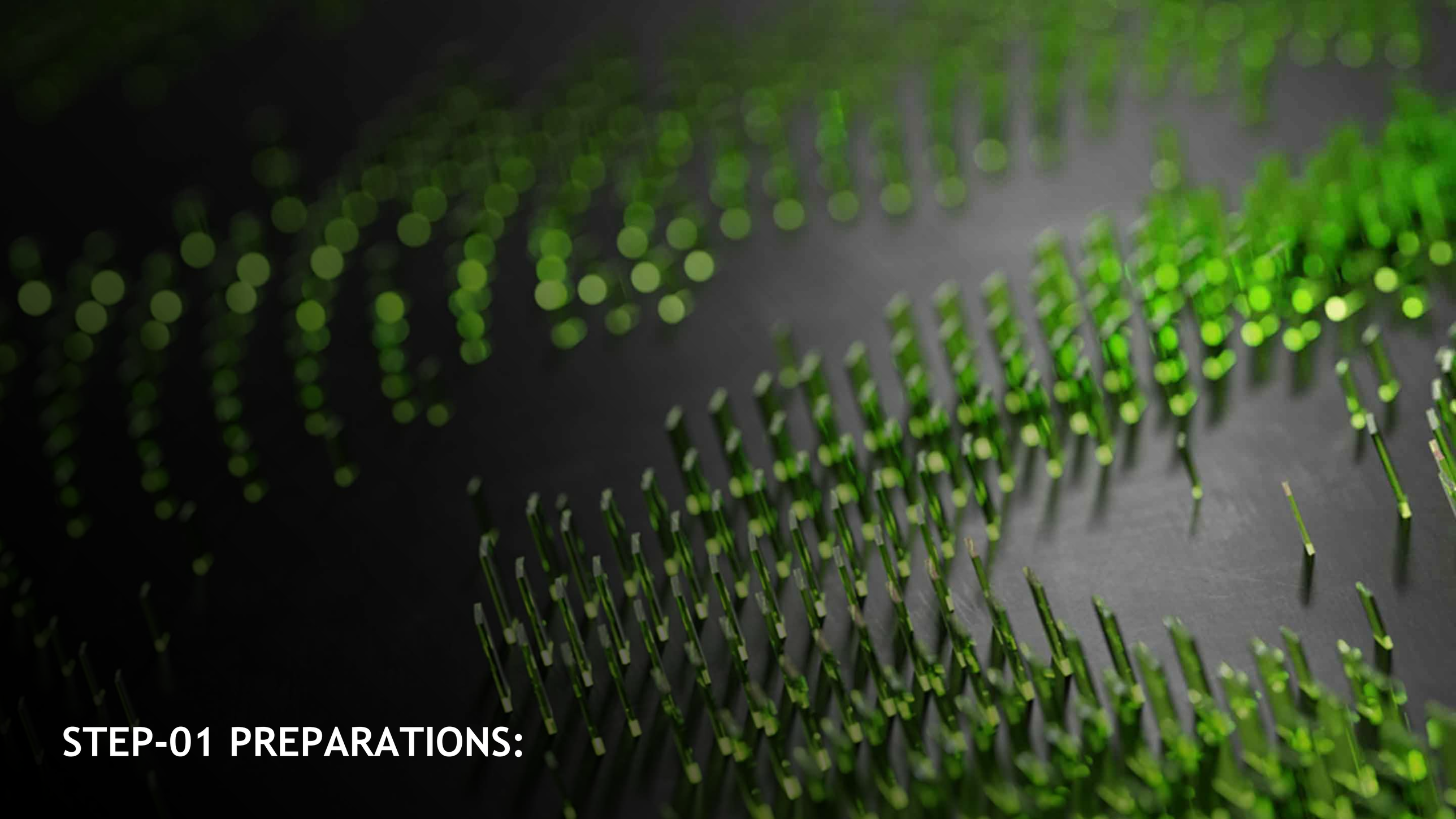
- Step-01: Preparing Ansible and Building Ansible playbooks
- Step-02: Disabling NCLU and enabling NVUE, Daemons, PTM
- Step-03: Layer 2 (Bridges, Access-Ports) and IP Addresses (host, lo)
- Step-04: Layer 3 (SVI, VRR, VRF's and BGP including EVPN)
- Step-05: Overlay (L2-EVPN-MH)
- Step-06: Overlay (L3-EVPN, VRF)
- Step-07: optional for reference (roles, templates, and vars)

AI-Infrastructure in 5 Minutes: Step-08
rebuilding nodes or setup a new simulation from the marketplace and run the playbook
(step-08 combining items from step-02 to step-06)

Next possible steps:
rewrite step-08 with concepts shown in step-06c and step-07 and/or verify PRA shared via Gitlab (Production-Ready-Automation)

NVIDIA

ALQ+TL



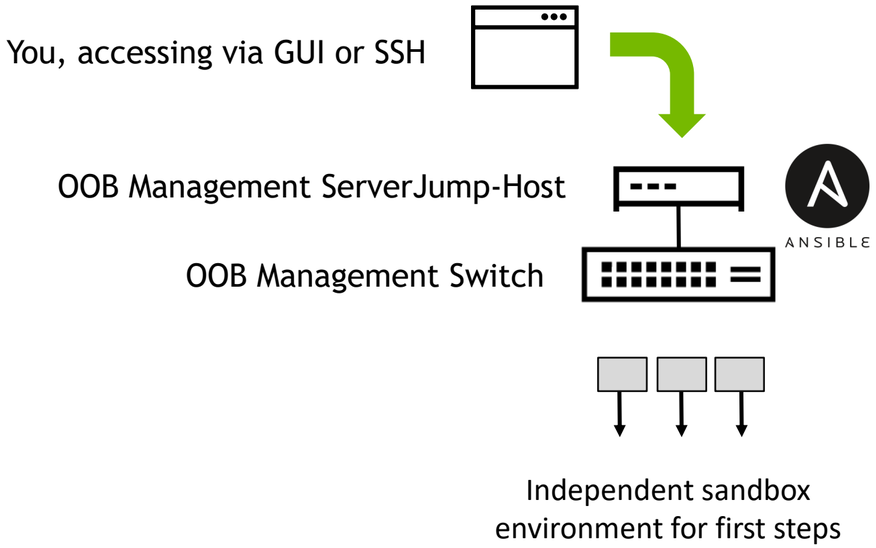
STEP-01 PREPARATIONS:

ZERO-TO-EVPN

Step-01 (Getting started with Ansible)

Create inventory

Create a playbook



TL

LAB AND/OR BREAK TIME

Step-01: 10 min

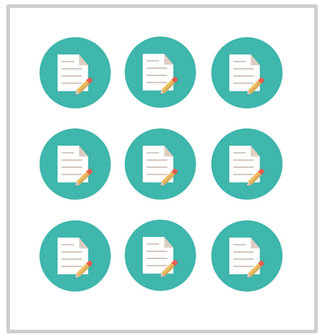
after the lab

ENVIRONMENT INTRODUCTION:

ZERO-TO-EVPN

The basic concept of step-01

- getting used to the files and structure
- working with host files and ad-hoc commands



repository

copy
clone
pull

oob-mgmt-
server

```
cumulus@oob-mgmt-server:~/ON-15$ tree
.
├── README.md
├── ansible.cfg
├── inventory
│   └── files
│       ├── daemons
│       ├── hdaemons
│       ├── hosts
│       ├── leaf01-after-step03
│       ├── leaf01-after-step04
│       ├── leaf02-after-step03
│       ├── leaf02-after-step04
│       ├── leaf03-after-step03
│       ├── leaf04-after-step03
│       ├── leaf05-after-step03
│       ├── leaf06-after-step03
│       ├── server11
│       ├── server12
│       ├── server13
│       ├── spine01-after-step03
│       ├── spine01-after-step04
│       ├── spine02-after-step03
│       ├── storage01
│       ├── topology.dot
│       ├── vm01
│       └── vm02
├── play-step-02.sh
├── play-step-03.sh
├── play-step-04.sh
├── play-step-05-l2.sh
├── play-step-05-mh.sh
├── play-step-06.sh
├── play-step-06b.sh
├── play-step-06c.sh
├── step-02
│   ├── prepare_hypervisor
│   │   ├── hypervisor01-if
│   │   ├── ifupdown2.conf
│   │   ├── main.yml
│   │   └── vswitches.sh
│   ├── prepare_switches
│   │   └── main.yml
│   └── prepare_vm
│       └── main.yml
<SNIP>
```

/inventory/files

Could be used as a “shopping cart” or kitchen-sink to hold items

1) Config files
2) Controller scripts
3) Orchestration items

ZERO-TO-EVPN

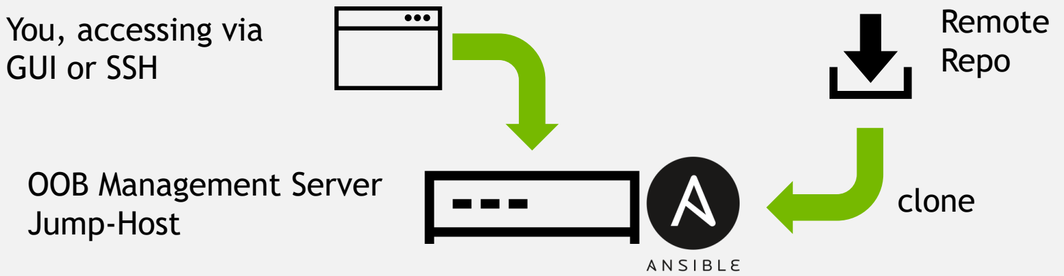
Easy Linux Networking

cumulus@oob-mgmt-server: $ git clone https://github.com/laquiante/ON-15

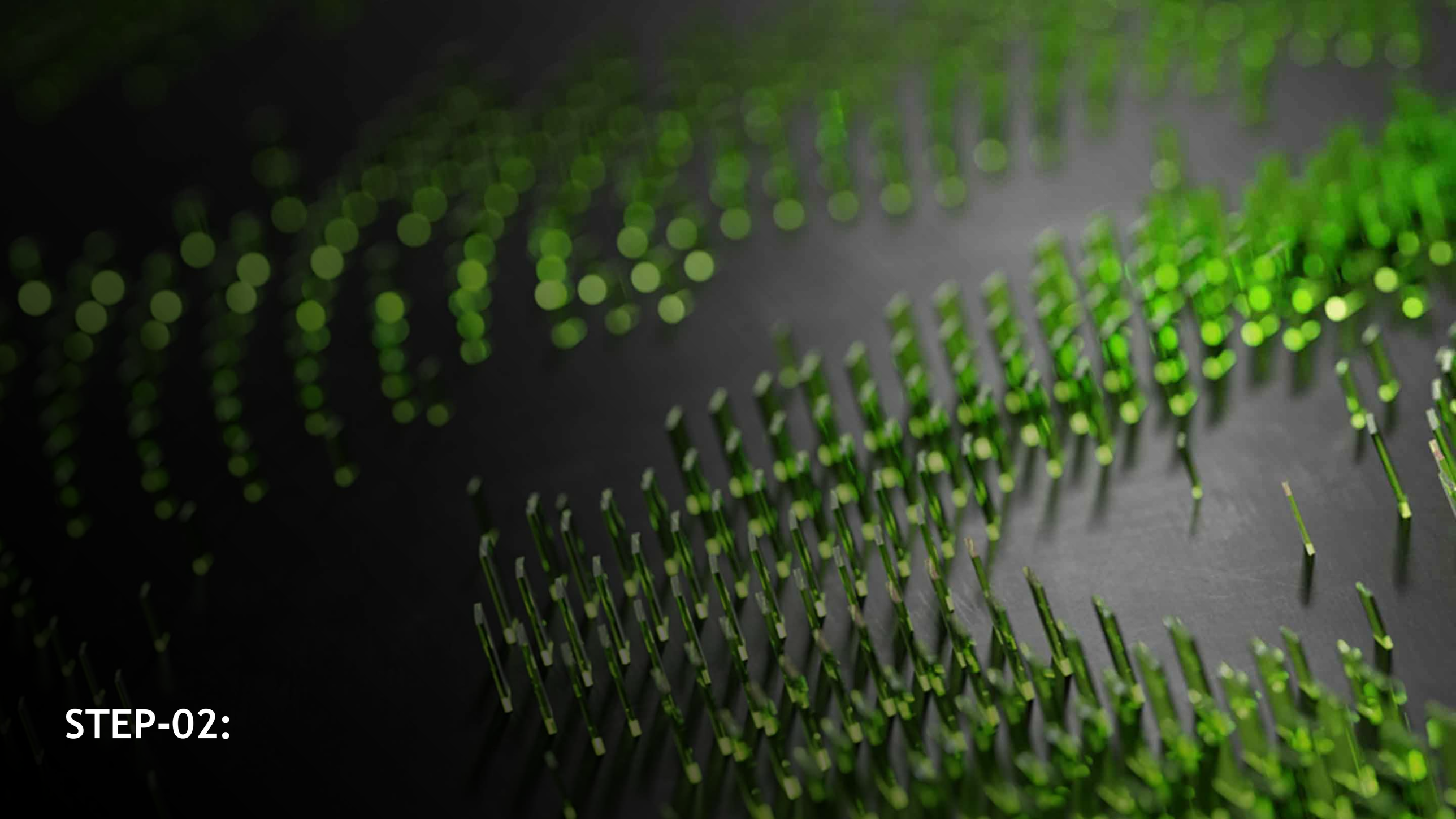
/etc/ansible/hosts

/home/cumulus/ON-15/ansible.cfg

/home/cumulus/ON-15/...



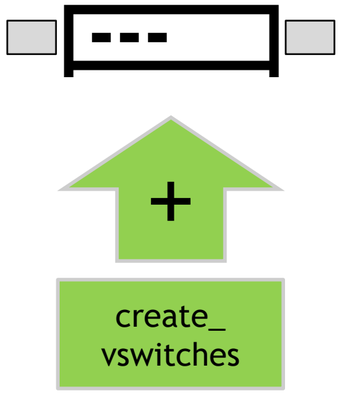
ALQ


STEP-02:

ZERO-TO-EVPN

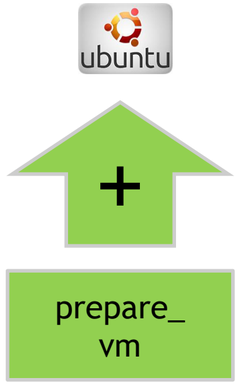
The basic concept of step-02
- preparing the infrastructure

hypervisor01



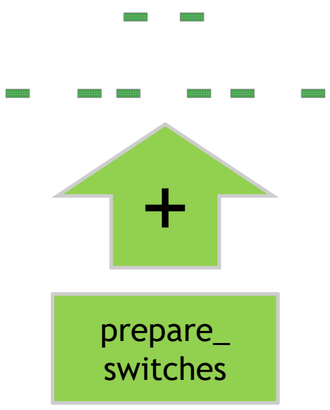
create_
vswitches

vm[01:02]

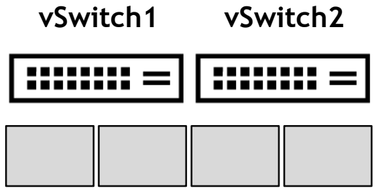


prepare_
vm

spine[01:02]
leaf[01:06]



prepare_
switches



Install routing on host

Switch NCLU to NVUE
Prepare FRR
Setup PTM

PTM : prescriptive topology manager
NVUE: NVIDIA user experience (starting CL 5.0 GA)
NCLU: Network Command Line Utility (up to CL 4.4.x)

ALQ

ZERO-TO-EVPN

Easy Linux Networking

Step-02: Disabling NCLU and enabling NVUE, Daemons, PTM…

was ip(route2) based

prepare_ hypervisor	/	create_ vswitches		
prepare_ vm	/	install_ FRR	use_ hdaemons	
prepare_ switches	/	enable_ NVUE	use_ daemons	enable_ PTM

Each playbook is called main.yml

The foundation here is mostly just simple Ansible + Linux

Note: We use FRR on both networking and compute nodes.
There might be a difference in version or product, thus we use in this workshop dedicated routing “configuration files == daemon files, daemon vs. hdaemon” to allow individual setup.

ALQ

NVIDIA

ZERO-TO-EVPN

Easy Linux Networking

```
cumulus@oob-mgmt-server:~/ON-15$
```

play-step-02-reference-hypervisor-vms-switches.sh
play-step-02-student-lab.sh

Each step comes with two options:
- verify/configure the "student" track
- run the "reference script

```
3 lines (3 sloc)  290 Bytes
1  ansible-playbook -i /home/cumulus/ON-15/inventory/files/hosts ./step-02/prepare_hypervisor/main.yml
2  ansible-playbook -i /home/cumulus/ON-15/inventory/files/hosts ./step-02/prepare_vm/main.yml
3  ansible-playbook -i /home/cumulus/ON-15/inventory/files/hosts ./step-02/prepare_switches/main.yml
```

~/ON-15/inventory/files/hosts

prepare_ hypervisor	/	create_ vswitches		
prepare_ vm	/	install_ FRR	use_ hdaemons	
prepare_ switches	/	enable_ NVUE	use_ daemons	enable_ PTM

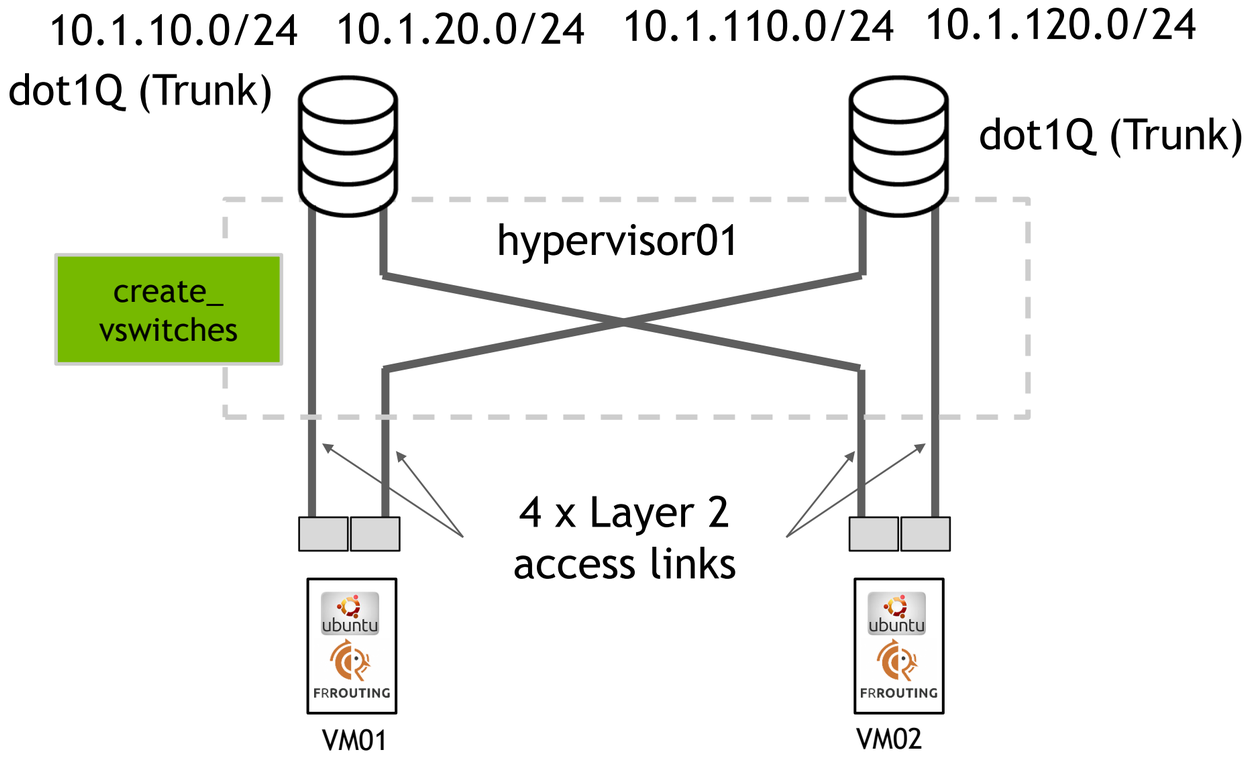
Each playbook is called main.yml

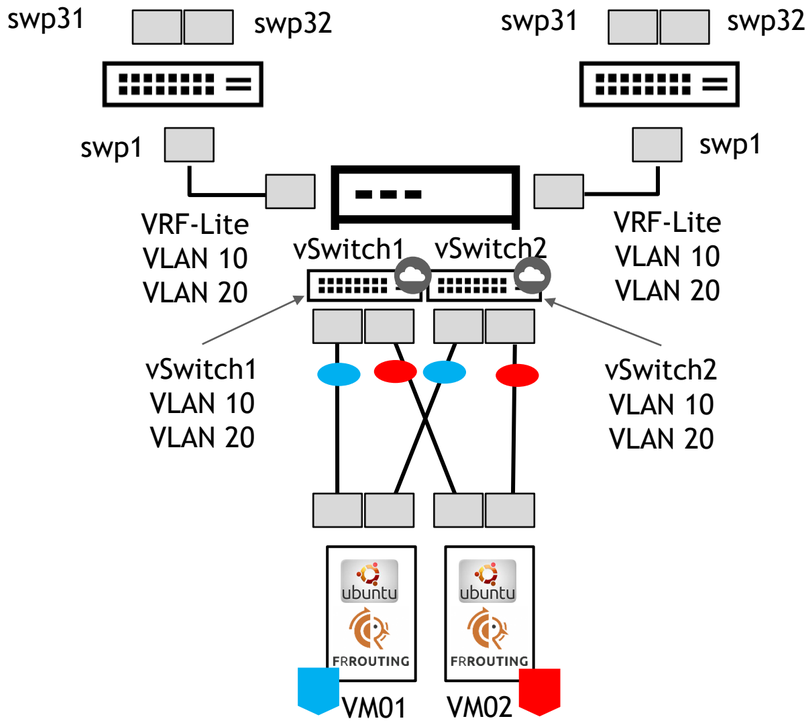
ALQ

STEP-02

Prepare the hypervisor via e/n/I file (previously this step used iproute2)

The RoH (FRR on VM01 and VM02) has been added to just show the concept.





Note: iproute2 commands via shell script “vSwitches.sh” are commented out in the playbook but left in draft state for reverence and inspiration.

ALQ

LAB AND/OR BREAK TIME

Step-02: 10 min

ZERO-TO-EVPN

How to verify what is "really" needed going forward

enable_ NVUE

On any switch commands starting with $ nv … should work e.g.,

```
cumulus@leaf01:mgmt:~$ nv show platform hardware --operational
            operational        description
----------  -----------------  ----------------------------------------
model       vx                 The platform's model identifier
system-mac  44:38:39:00:00:4a  The MAC provided by eeprom for system-mac
vendor      cumulus            The platform's vendor
```

use_ daemons

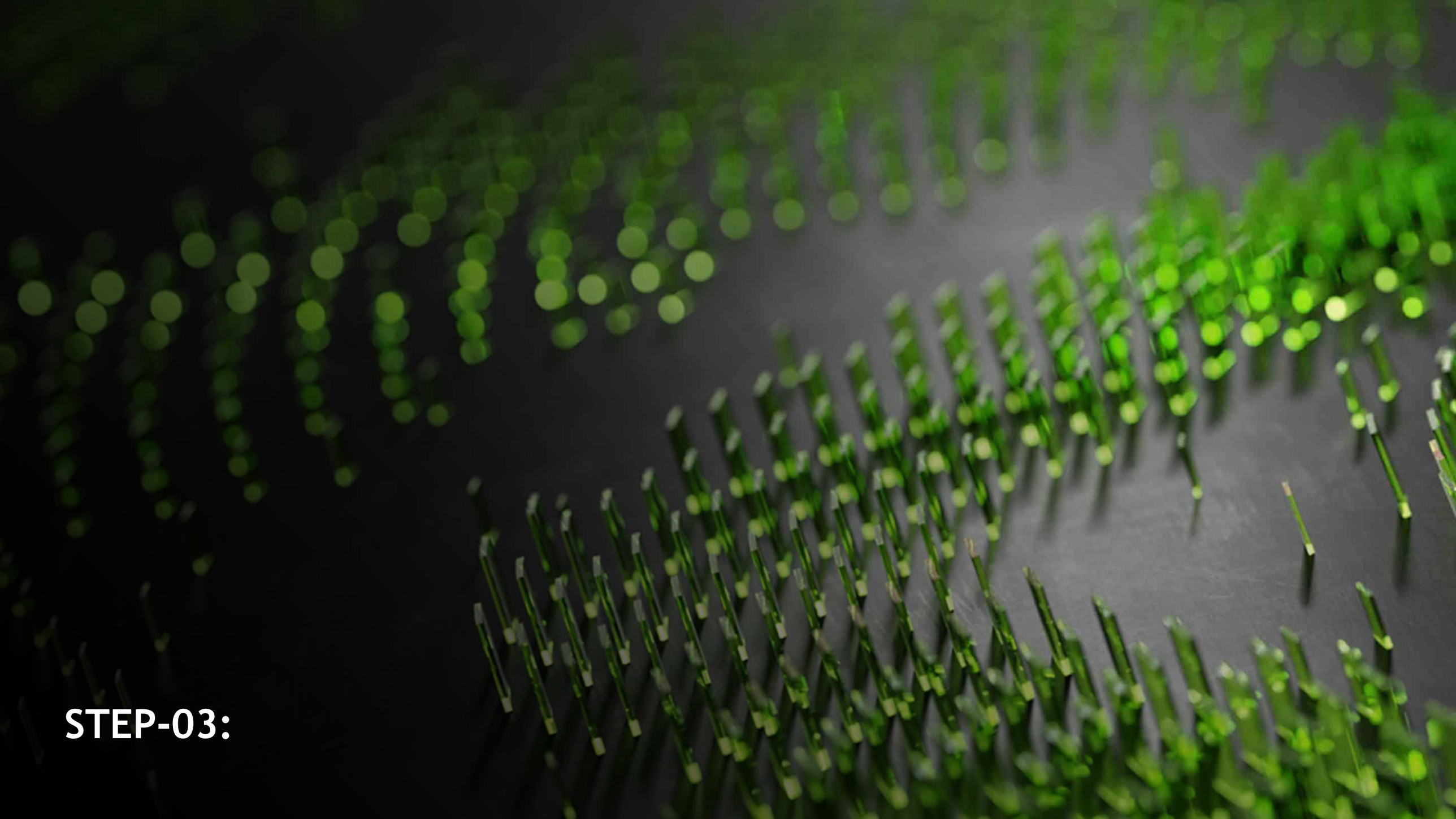
On any switch the file /etc/frr/daemons should show bgpd=yes

```
bgpd=yes
ospfd=no
ospf6d=no
ripd=no
ripngd=no
isisd=no
fabricd=no
pimd=no
ldpd=no
nhrpd=no
eigrpd=no
babeld=no
sharpd=no
pbrd=no
fabricd=no
vrrpd=no
```

In case of need just run the reference script

```
cumulus@oob-mgmt-server:~/ON-15$
play-step-02-reference-...
```

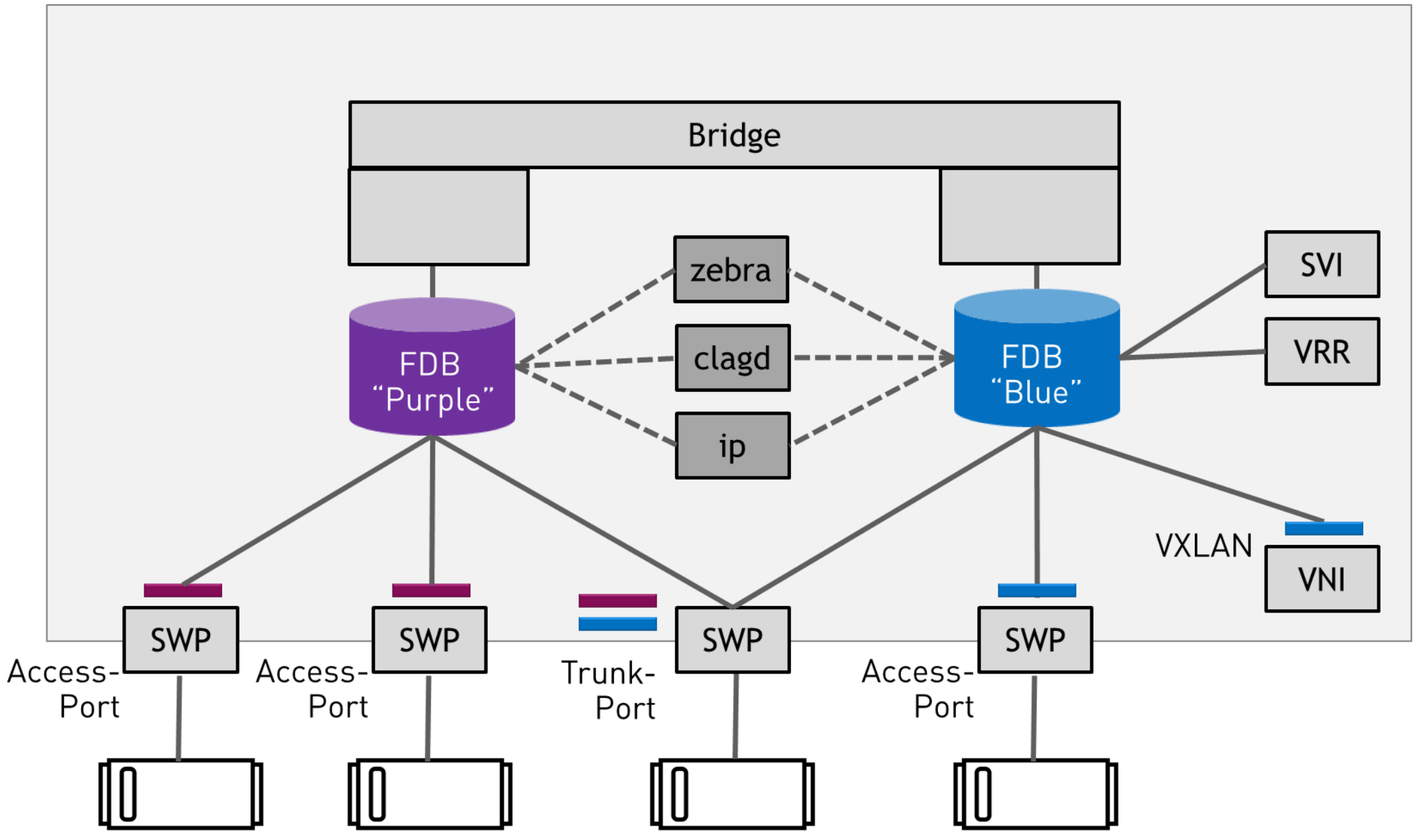
TL


STEP-03:

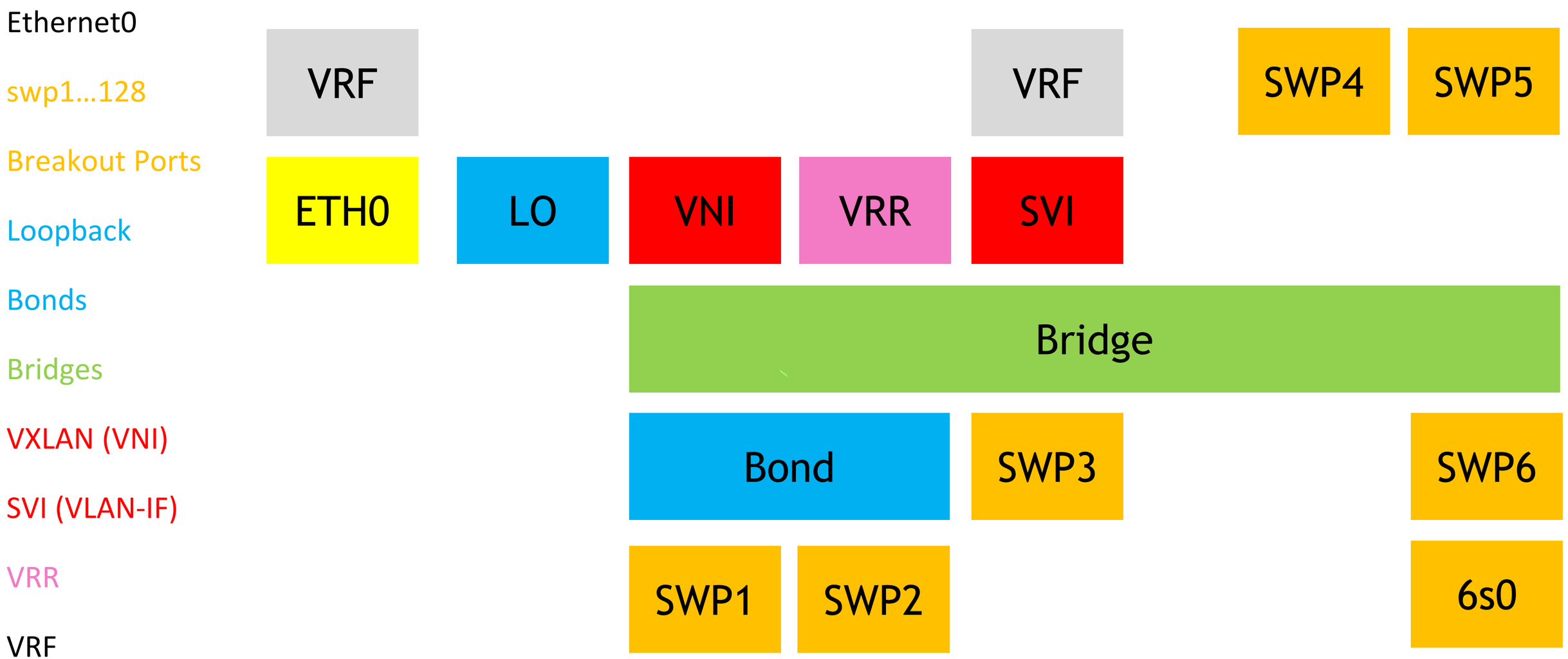
ZERO-TO-EVPN

The basic concept of step-03

Step-03: Layer 2 (Bridges, Access-Ports) and IP Addresses (host, lo)



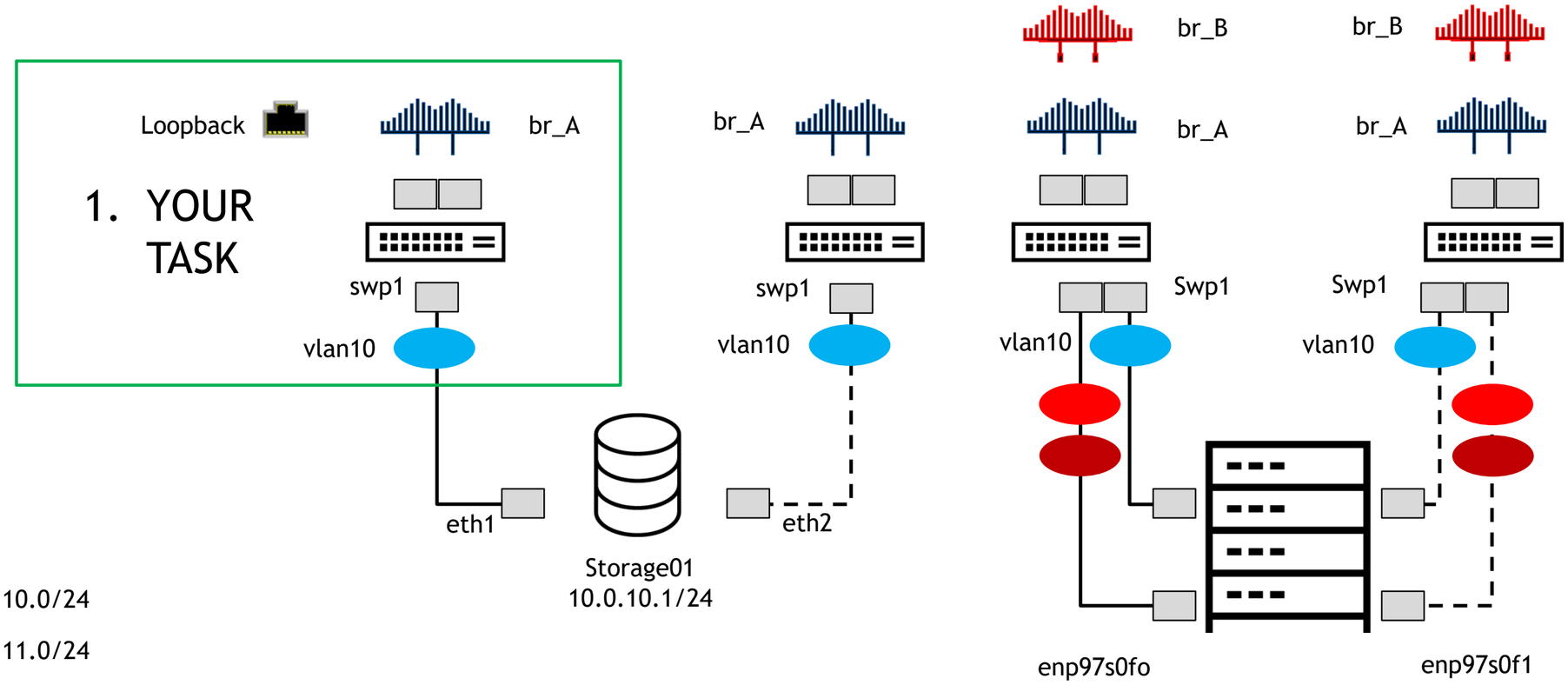
BRIDGE AND INTERFACES



ZERO-TO-EVPN

Easy Linux Networking

Step-03: Layer 2 (Bridges, Access-Ports) and IP Addresses (host, lo)



PREPARATION

Information source: Files, DB, IPAM, Notes,…

L3 Structure:

Anycast Addresses		: a.b.c.254/24
SVI		: a.b.c.240-253/24
Reserved:		: a.b.c.1-9/24
Clients:		: a.b.c.10-199/24
Reserve:		: a.b.c.200-239/24
Multicast (BUM):		: 239.0.0.VLAN-ID
Storage01:		: 10.0.10.1/24
Server11:		: 10.0.10.11/24
Server12:		: 10.0.20.12/24
		: 10.0.10.12/24
Server13:		: 10.0.11.13/24
VM01:		: 10.1.10.101/24
		: 10.1.120.101/24
VM02:		: 10.1.20.102/24
		: 10.1.120.102/24
Leaf05	swp 1.10	: 10.1.10.205/24
Leaf05	swp 1.20	: 10.1.20.205/24
Leaf06	swp 1.10	: 10.1.110.206/24
Leaf06	swp 1.20	: 10.1.120.206/24

DB (repository)

ANS & Loopbacks:

Spine01:	4200000201	: 192.168.0.201/32
Spine02:	4200000202	: 192.168.0.202/32
Leaf01:	4200000001	: 192.168.0.1/32
Leaf02:	4200000002	: 192.168.0.2/32
Leaf03:	4200000003	: 192.168.0.3/32
Leaf04:	4200000004	: 192.168.0.4/32
Leaf05:	4200000005	: 192.168.0.5/32
Leaf06:	4200000006	: 192.168.0.6/32
VM01:	4200000011	: 192.168.0.101/32
VM02:	4200000012	: 192.168.0.102/32

main ON-15 / inventory / files /

laquiante Create vm02

..

daemons

hdaemons

hosts

server11

server12

server13

storage01

topology.dot

vm01

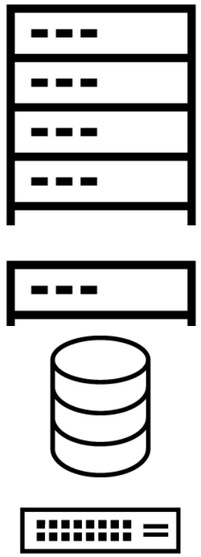
vm02

1. E/N/I

1. E/N/I

standard modules

copy



ALQ

NVIDIA

PREPARATION

Information source: Files, DB, IPAM, Notes,…

L3 Structure:

Anycast Addresses		**: a.b.c.254/24**
SVI		: a.b.c.240-253/24
Reserved:		: a.b.c.1-9/24
Clients:		: a.b.c.10-199/24
Reserve:		: a.b.c.200-239/24
Multicast (BUM):	: 239.0.0.VLAN-ID	
Storage01:		**: 10.0.10.1/24**
Server11:		: 10.0.10.11/24
Server12:		: 10.0.20.12/24
		: 10.0.10.12/24
Server13:		: 10.0.11.13/24
VM01:		: 10.1.10.101/24
		: 10.1.120.101/24
VM02:		: 10.1.20.102/24
		: 10.1.120.102/24
Leaf05	swp 1.10	: 10.1.10.205/24
Leaf05	swp 1.20	: 10.1.20.205/24
Leaf06	swp 1.10	: 10.1.110.206/24
Leaf06	swp 1.20	: 10.1.120.206/24

ANS & Loopbacks:

Spine01:	4200000201	: 192.168.0.201/32
Spine02:	4200000202	: 192.168.0.202/32
Leaf01:	4200000001	: 192.168.0.1/32
Leaf02:	4200000002	: 192.168.0.2/32
Leaf03:	4200000003	: 192.168.0.3/32
Leaf04:	4200000004	: 192.168.0.4/32
Leaf05:	4200000005	: 192.168.0.5/32
Leaf06:	4200000006	: 192.168.0.6/32
VM01:	4200000011	: 192.168.0.101/32
VM02:	4200000012	: 192.168.0.102/32

1. STORAGE01

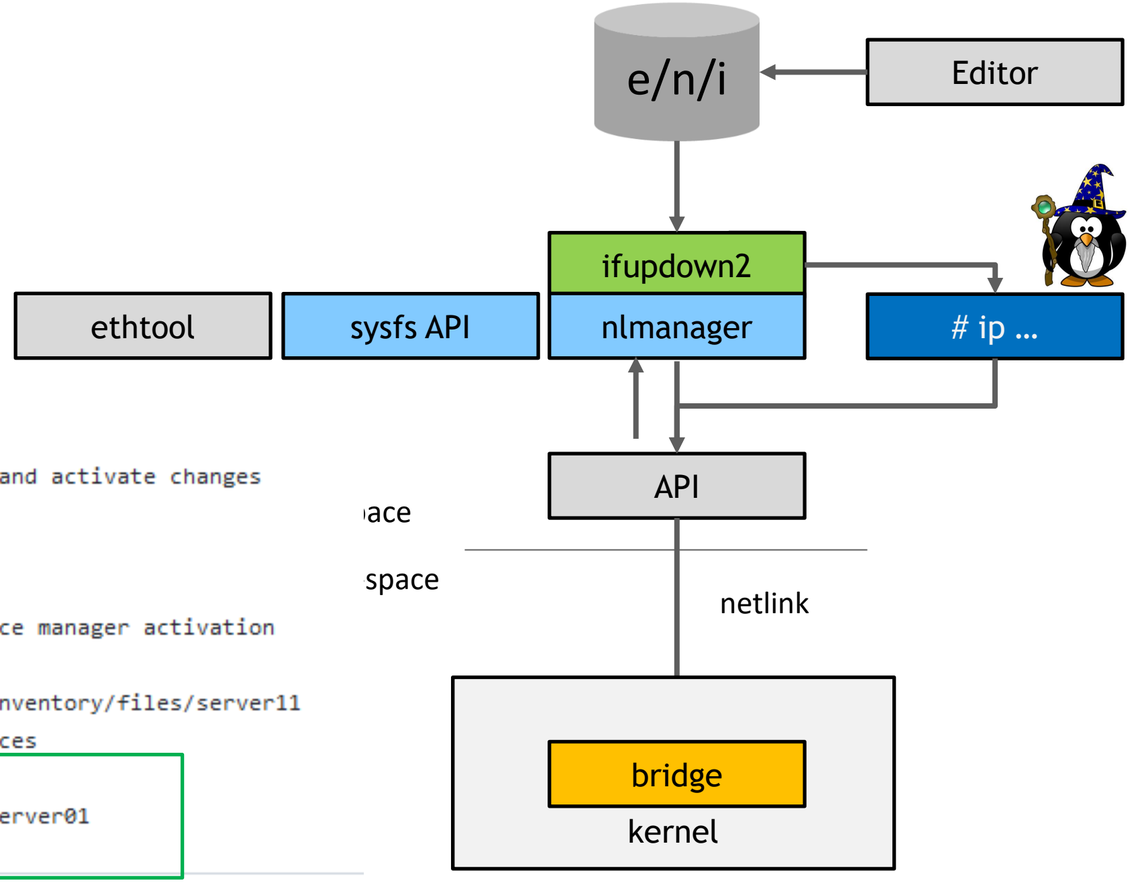
```
cumulus@storage01:~$ cat /etc/network/interfaces
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet dhcp
    post-up sysctl -w net.ipv6.conf.eth0.accept_ra=2

auto eth1
iface eth1 inet static
  bond-master uplink

  auto eth2
  iface eth2 inet static
    bond-master uplink

auto uplink
iface uplink inet static
  address 10.0.10.1
  netmask 255.255.255.0
  mtu 9216
  bond-slaves eth1 eth2
  bond-mode 802.3ad
  bond-miimon 100
  bond-lacp-rate 1
  bond-min-links 1
  bond-xmit-hash-policy layer3+4
  post-up ip route add 10.0.0.0/8 via 10.0.10.254
```

INTERFACE MANAGER & MAGIC

How it works and how to non-disruptively update the interfaces

1. IFUPDOWN2

```
---
- hosts: server11
  name: create bond, set loopback and activate changes
  become: yes
  gather_facts: no
  tasks:
    - name: e/n/i file and interface manager activation
      copy:
        src: /home/cumulus/ON-15/inventory/files/server11
        dest: /etc/network/interfaces
    # ifreload
    - name: activate changes for Server01
      shell: /sbin/ifreload -a
```



ALQ

LAB AND/OR BREAK TIME

Step-03: 15 min

ALQ/TL

ZERO-TO-EVPN

How to verify what is "really" needed going forward

lo

Verify on leaf01 if the interface loopback lo uses the correct IPv4 address

```
cumulus@leaf01:mgmt:~$ ip -br -4 a
lo               UNKNOWN        127.0.0.1/8 192.168.0.1/32
eth0             UP             192.168.200.9/24
```

br-A

Verify on leaf01 if the newly configured bridge br_A has learned the MAC address of storage01

Storage01:
44:38:39:00:00:01

```
cumulus@leaf01:mgmt:~$ bridge fdb
44:38:39:00:00:01 dev swp1 vlan 10 master br_A
44:38:39:00:00:02 dev swp1 master br_A permanent
44:38:39:00:00:02 dev swp1 self permanent
```

PTM

Optionally verify that leaf01 shows lldp connectivity to spine01 and spine02. (Storage01 LLDP uses an unexpected value and will show PTM fail.

```
cumulus@leaf01:mgmt:~$ sudo ptmctl
----------------------------------------
port   cbl    BFD    BFD   BFD    BFD
     status  status  peer  local  type
----------------------------------------
swp1   fail   N/A    N/A   N/A    N/A
swp31  pass   N/A    N/A   N/A    N/A
swp32  pass   N/A    N/A   N/A    N/A
```

In case of need just run the reference script

```
cumulus@oob-mgmt-server:~/ON-15$
play-step-03-reference....sh
```

NVIDIA

TL

SUMMARY

& next steps

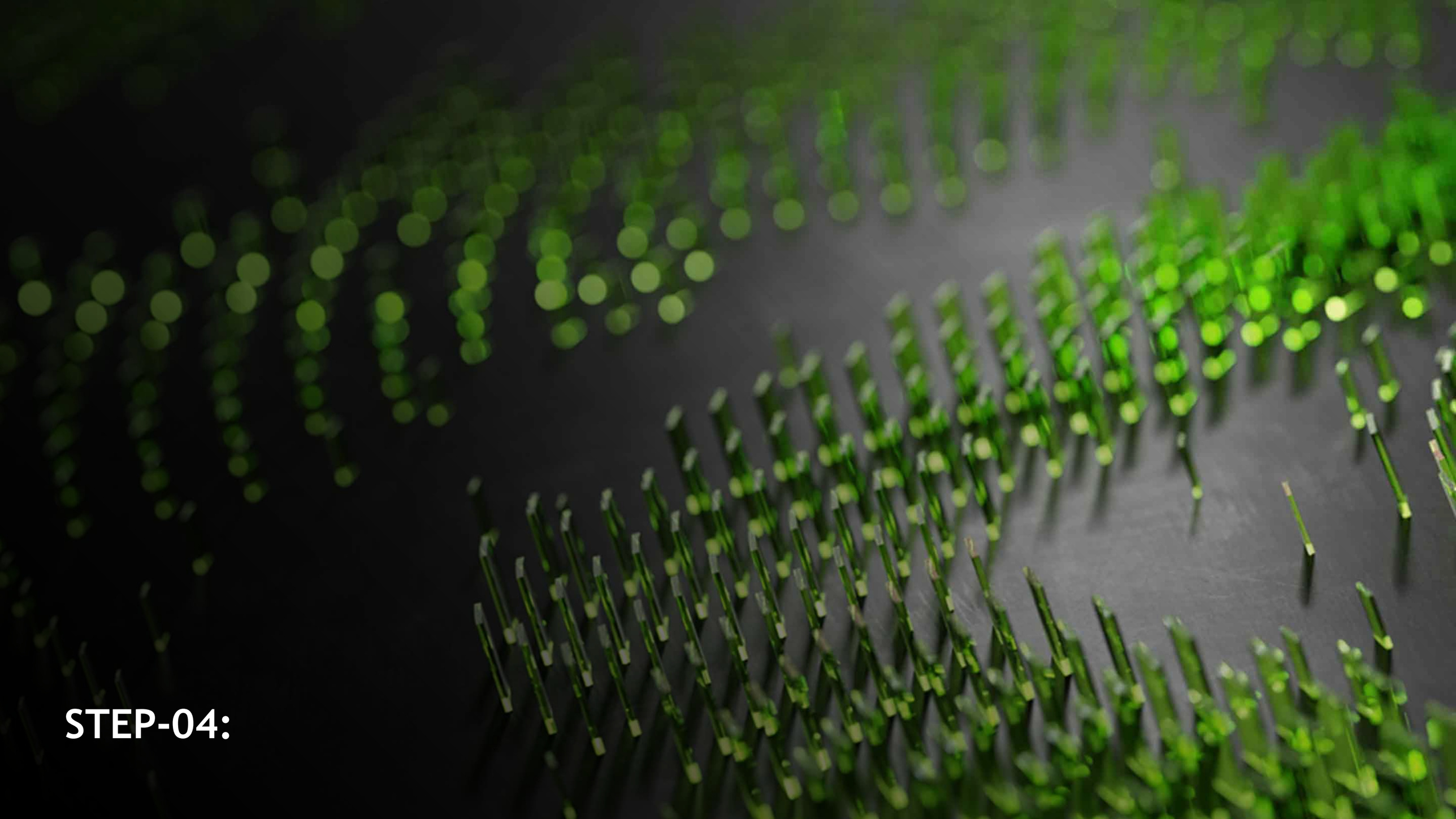
Ok, layer 2 is working but with unique testing points (SVI) in the network the nodes could verify more.

With layer 3 we could offer connectivity between vlans (vlan 10 and 11) and to offer mobility we need the same gateway on all Top-of-Rack (TOR) switches aka leafs.

With layer 3 in place, we need immediately isolation (VRFs) to support tenants (Tenant_A and Tenant_B).

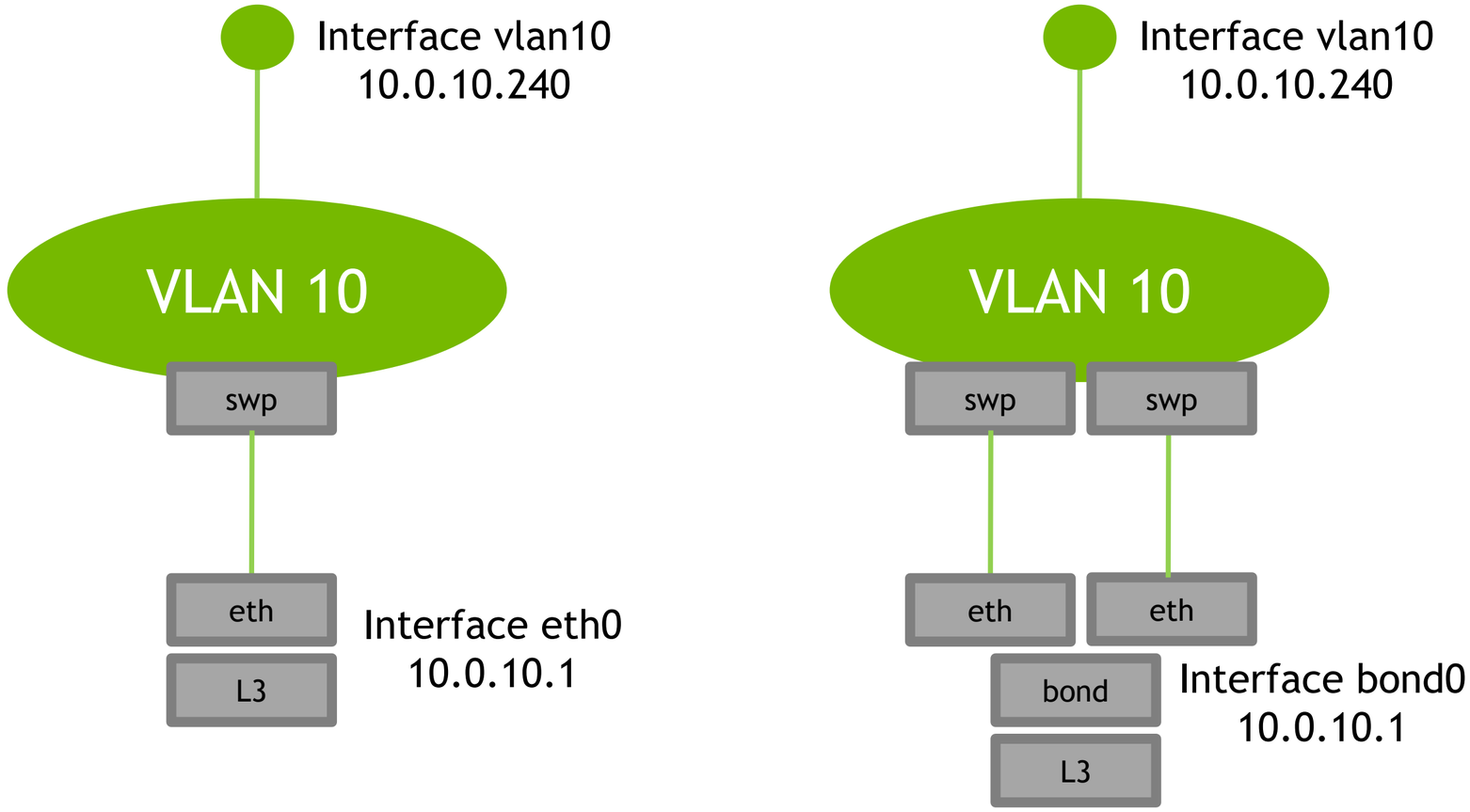
And last but not least we would benefit from easy dynamic routing e.g. BGP.

This leads us to step-04


STEP-04:

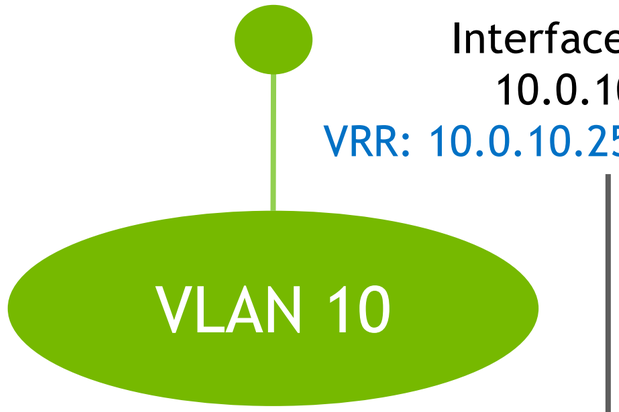
ZERO-TO-EVPN

The basic concept of step-04

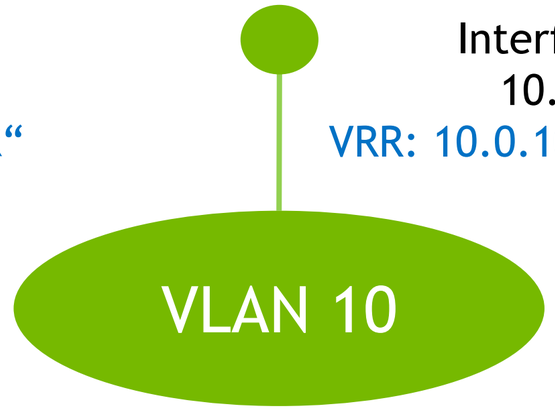


ZERO-TO-EVPN

The basic concept of step-04

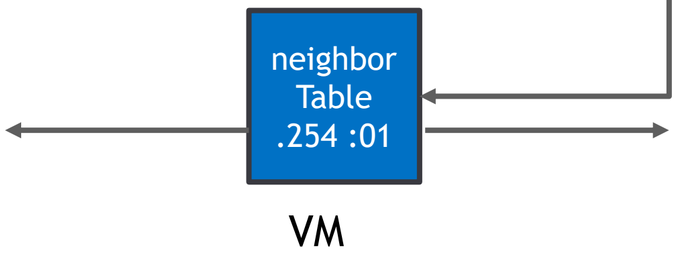


Interface vlan10
10.0.10.240
VRR: 10.0.10.254+MAC „VRR“



Interface vlan10
10.0.10.240
VRR: 10.0.10.254+MAC “VRR”

```
e/n/i
post-up ip route add 10.0.0.0/8 via 10.0.10.254
```



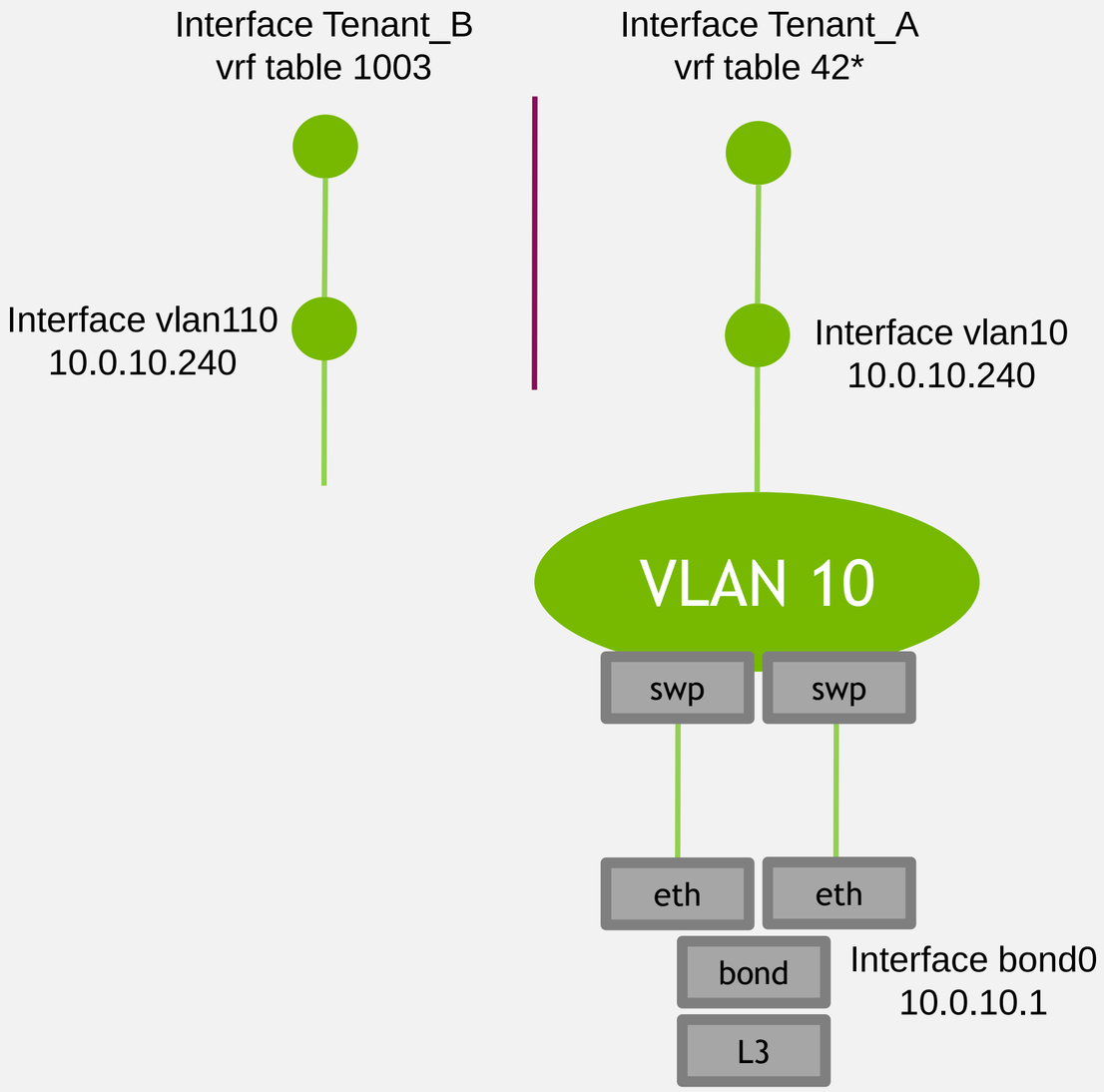

```
cumulus@storage01:~$ ip -4 nei
10.0.10.243 dev uplink lladdr 44:38:39:00:00:06 STALE
192.168.200.1 dev eth0 lladdr 44:38:39:00:00:36 REACHABLE
10.0.10.24 dev uplink  FAILED
192.168.200.250 dev eth0  INCOMPLETE
10.0.10.241 dev uplink lladdr 44:38:39:00:00:04 STALE
10.0.10.244 dev uplink lladdr 44:38:39:00:00:08 STALE
10.0.10.254 dev uplink lladdr 00:00:5e:00:00:01 STALE
10.0.10.240 dev uplink lladdr 44:38:39:00:00:02 STALE
10.0.10.11 dev uplink lladdr 44:38:39:00:00:05 STALE
```

ALQ

ZERO-TO-EVPN

The basic concept of step-04

- Linux table numbers for VRF’s are more flexible than under Cumulus Linux



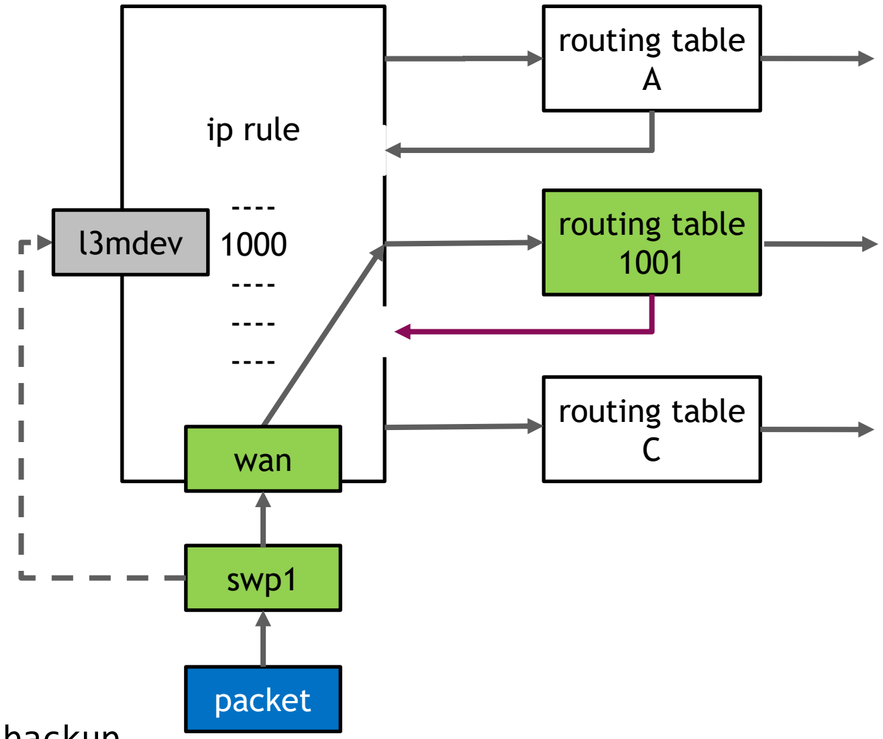
VRF ROUTING TABLE

A closer look (customer example)

```
cumulus@spine01:~$ ip route show vrf wan
1.1.1.0/30 dev swp1 proto kernel scope link src 1.1.1.2
10.10.1.0/24 nhid 62 via 1.1.1.1 dev swp1 proto bgp metric 20
20.20.2.0/24 dev vlan20 proto kernel scope link src 20.20.2.254
```

The solution against table leaking:

```
cumulus@provider01:mgmt:~$ ip route show vrf wan
unreachable default metric 4278198272
1.1.1.0/30 dev swp1 proto kernel scope link src 1.1.1.1
10.10.1.0/24 dev vlan10 proto kernel scope link src 10.10.1.254
```

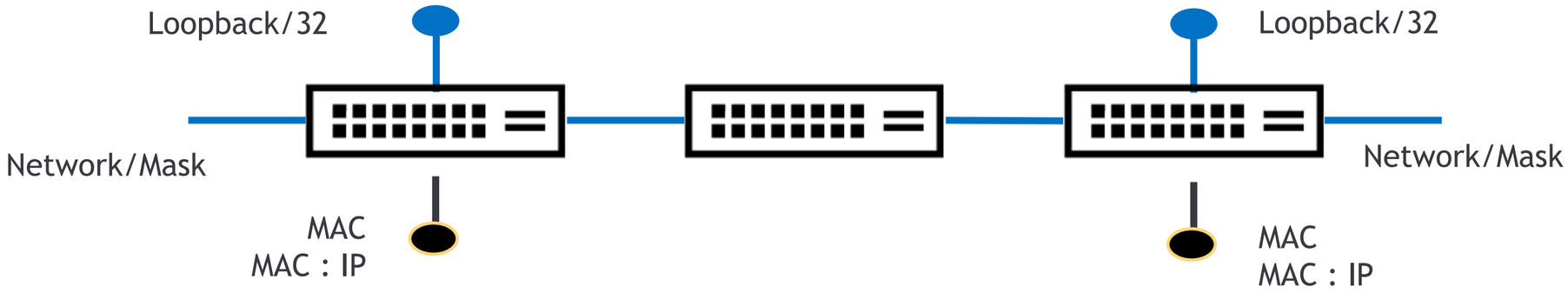



```
cumulus@provider01:mgmt:~$ net show route vrf wan
show ip route vrf wan
=====================
Codes: K - kernel route, C - connected, S - static, R - RIP,
       O - OSPF, I - IS-IS, B - BGP, E - EIGRP, N - NHRP,
       T - Table, v - VNC, V - VNC-Direct, A - Babel, D - SHARP,
       F - PBR, f - OpenFabric,
       > - selected route, * - FIB route, q - queued, r - rejected, b - backup
       t - trapped, o - offload failure
VRF wan:
K>* 0.0.0.0/0 [255/8192] unreachable (ICMP unreachable), 00:01:52
C>* 1.1.1.0/30 is directly connected, swp1, 00:01:52
C>* 10.10.1.0/24 is directly connected, vlan10, 00:01:52
```

reference

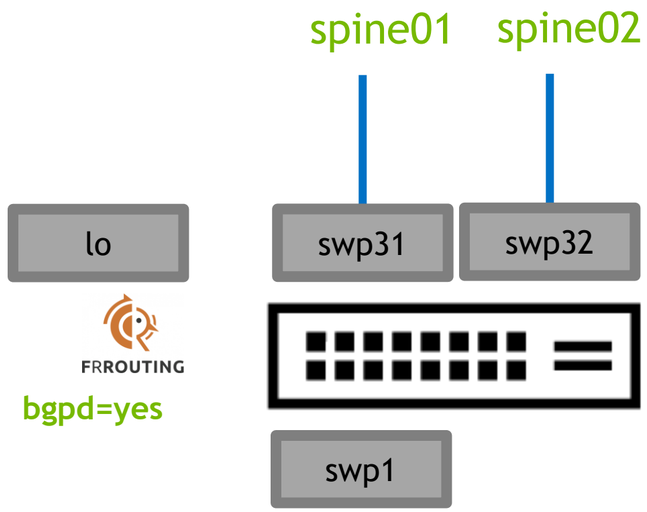
ZERO-TO-EVPN

The basic concept of step-04 (Underlay/Overlay)



ALQ

BGP EVPN MADE EASY

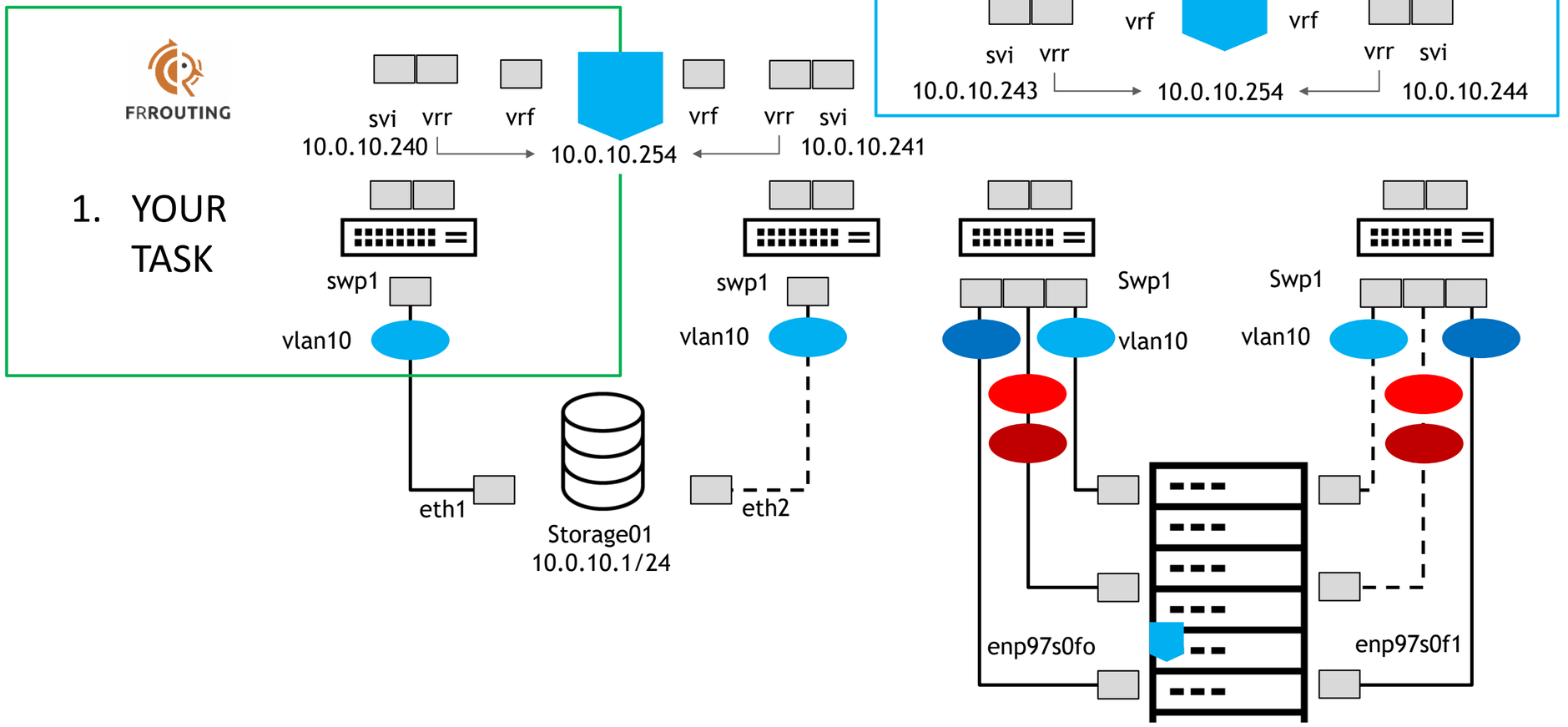



```
!
router bgp 4200000001
 bgp router-id 192.168.0.1
 bgp bestpath as-path multipath-relax
 neighbor swp31 interface remote-as external
 neighbor swp32 interface remote-as external
 !
 address-family ipv4 unicast
  network 192.168.0.1/32
 exit-address-family
 !
 address-family l2vpn evpn
  neighbor swp31 activate
  neighbor swp32 activate
  advertise-all-vni
 exit-address-family
!
```

ZERO-TO-EVPN

Easy Linux Networking

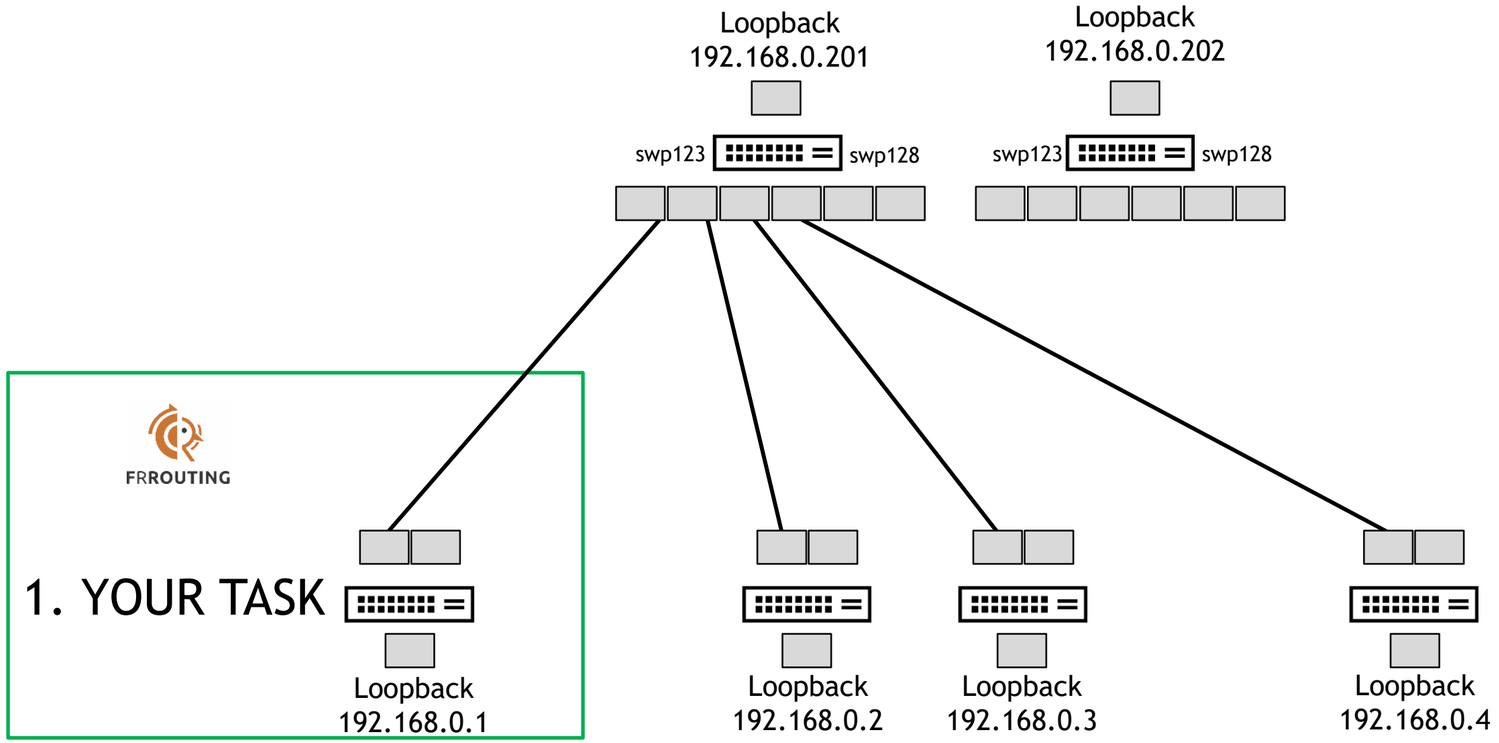
Step-04: Layer 3 (SVI, VRR, VRF's)
Step-04: Routing (BGP including EVPN preparation)



ZERO-TO-EVPN

Easy Linux Networking

Step-04: Layer 3 (SVI, VRR, VRF's)
Step-04: Routing (BGP including EVPN preparation)



LAB AND/OR BREAK TIME

Step-04: 15 min

ALQ/TL

ZERO-TO-EVPN

How to verify what is "really" needed going forward

Routing table

Verify on leaf01 the routing table for VRF Teanant_A

```
cumulus@leaf01:mgmt:~$ ip route show vrf Tenant_A
unreachable default metric 4278198272
10.0.10.0/24 dev vlan10 proto kernel scope link src 10.0.10.240
10.0.10.0/24 dev vlan10-v0 proto kernel scope link src 10.0.10.254 metric 1024
```

```
cumulus@leaf01:mgmt:~$ ip nexthop
id 15 dev lo scope host proto zebra
id 16 dev eth0 scope host proto zebra
id 18 dev vlan10 scope host proto zebra
id 19 dev eth0 scope link proto zebra
id 22 dev swp31 scope link proto zebra
id 23 dev vlan10-v0 scope link proto zebra
id 25 via 192.168.200.1 dev eth0 scope link proto zebra
id 26 blackhole proto zebra
id 27 blackhole proto zebra
id 28 blackhole proto zebra
id 32 via fe80::4638:39ff:fe00:1d dev swp31 scope link proto zebra
id 35 via fe80::4638:39ff:fe00:29 dev swp32 scope link proto zebra
```

Ping SVI & VRR

Verify on storage01 if the SVI and VRR addresses of leaf01 can be reached

```
cumulus@storage01:~$ ping -c 3 10.0.10.240
PING 10.0.10.240 (10.0.10.240) 56(84) bytes of data.
64 bytes from 10.0.10.240: icmp_seq=1 ttl=64 time=0.295 ms
64 bytes from 10.0.10.240: icmp_seq=2 ttl=64 time=0.289 ms
64 bytes from 10.0.10.240: icmp_seq=3 ttl=64 time=0.358 ms

--- 10.0.10.240 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2038ms
rtt min/avg/max/mdev = 0.289/0.314/0.358/0.031 ms
```

Dynamic routing

Verify on leaf01 the "main" or "default" routing table to show the remote loopback addresses

```
cumulus@leaf01:mgmt:~$ ip route show
192.168.0.2 nhid 35 proto bgp metric 20
192.168.0.3 nhid 32 proto bgp metric 20
192.168.0.4 nhid 35 proto bgp metric 20
192.168.0.5 nhid 32 proto bgp metric 20
192.168.0.6 nhid 35 proto bgp metric 20
192.168.0.201 nhid 32 proto bgp metric 20
192.168.0.202 nhid 35 proto bgp metric 20
```

In case of need just run the reference script

```
cumulus@oob-mgmt-server:~/ON-15$
play-step-04-reference....sh
```

NVIDIA

ALQ

SUMMARY

& next steps

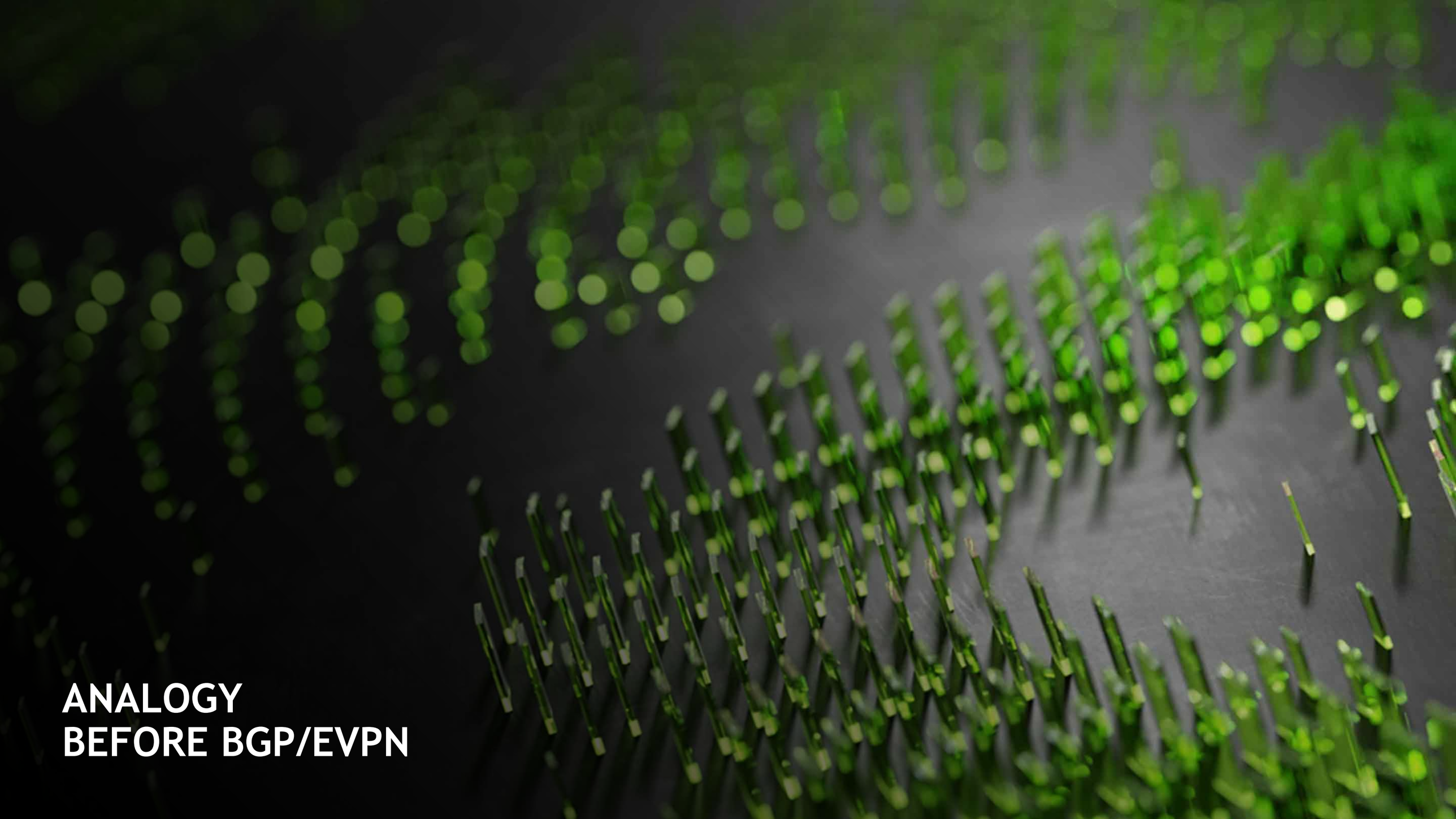
- Great job!

You created the infrastructure and now we can work on dynamic layer 2 tunnels.

- Next step:

We create Layer 2 connectivity (Storage01 to Server11) via the Layer 3 Infrastructure

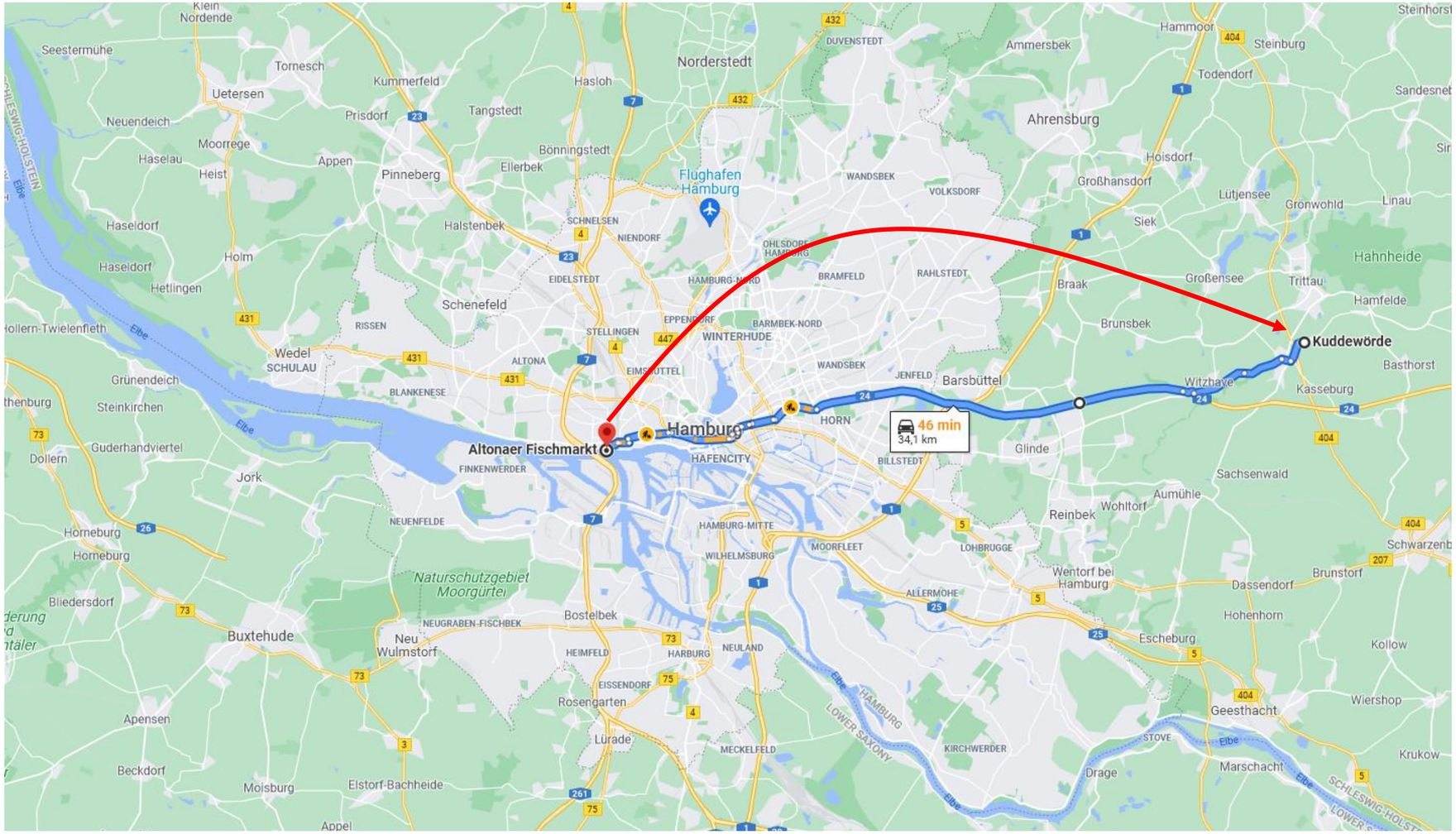
TL



ANALOGY
BEFORE BGP/EVPN

ANALOGY

Let’s assume we want to go from “Fischmarkt” to “Kuddewörde”

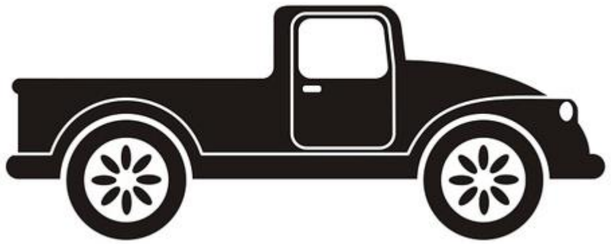


ALQ

ANALOGY

IP/TCP port 179

- We need a vehicle to get from "A" to "B" like from “Hamburg” to “Kuddewörde”
- This vehicle is “BGP”



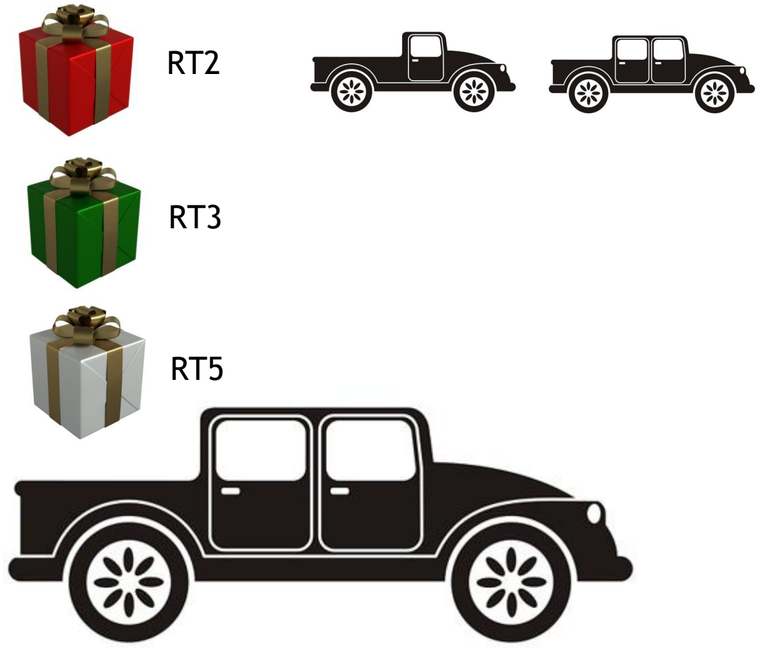
ALQ

ANALOGY

BGP – Address Family IPv4 Unicast

We like to carry our family, team, or IPv4 Loopback-Address with our vehicle.

Address-family : IPv4 Unicast



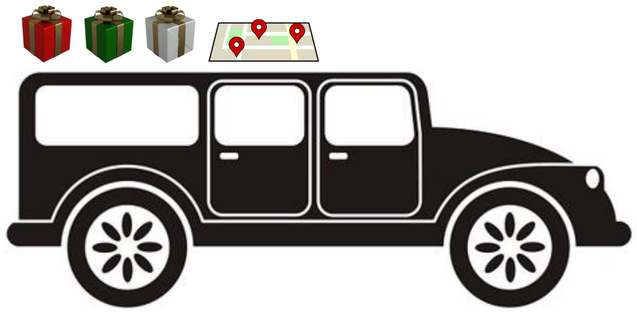
Our neighbors are traveling too, and they could buy an own vehicle.

Transportation wise, let us assume they deliver boxes labeled RT2, RT3, and RT5. More recently they are also carrying RT1 and RT4 boxes.

Address-family : L2VPN EVPN

ANALOGY

BGP – Address Family L2VPN EVPN



Since we are all good citizens, we don't mind sharing the ride with other families/teams or with those RT1-5 boxes.

Address-families :

IPv4 Unicast
L2VPN EVPN

IPv6 Unicast

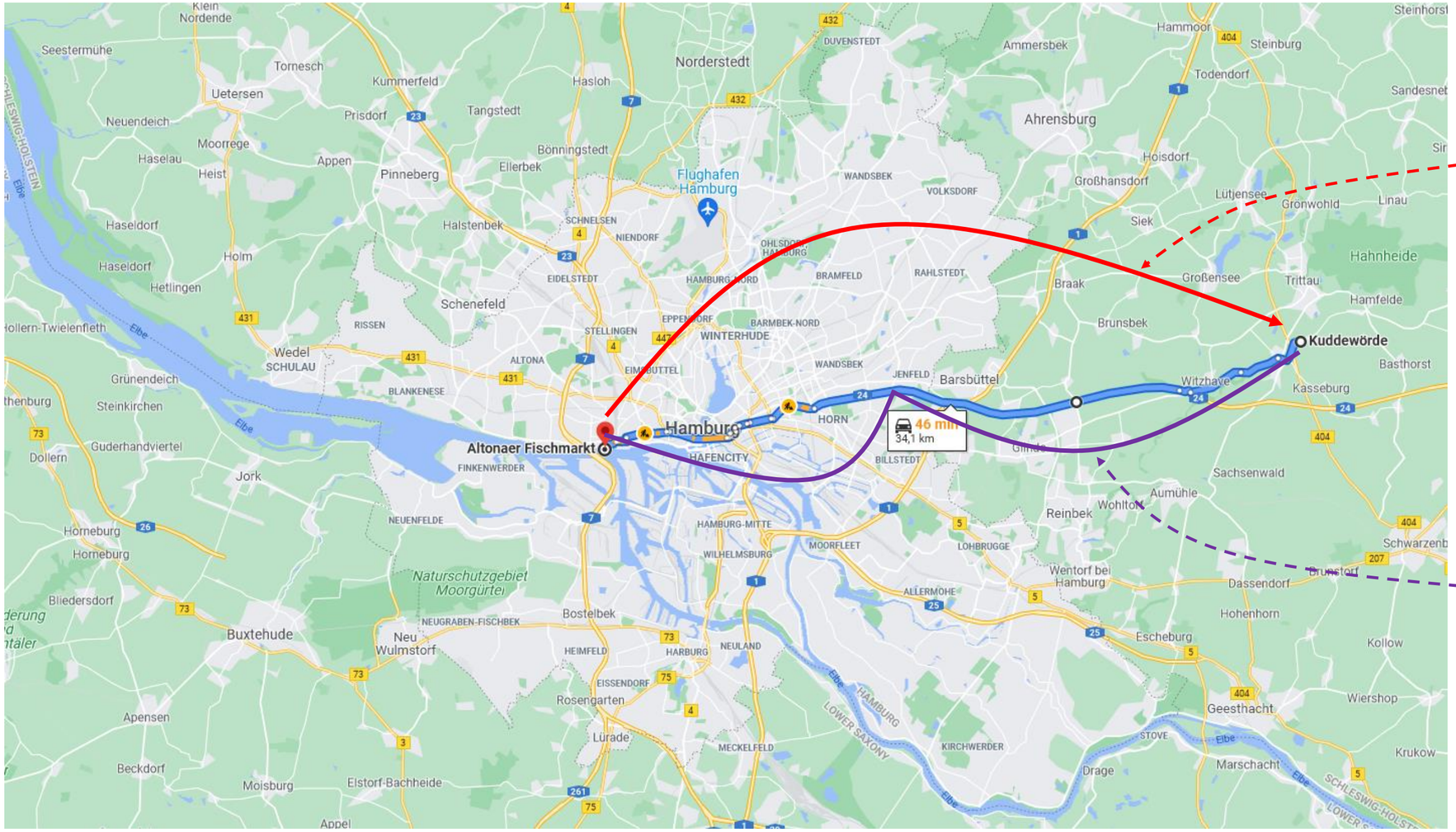
IPv4 Multicast
IPv6 VPN
….

Our vehicle MP-BGP (Multi-Protocol-BGP) looks more like a bus…but let's focus on two families today

ALQ

ANALOGY

Let’s assume we want to go from “Fischmarkt” to “Kuddewörde”



We need to decide, either:
Two vehicles,
one goes straight to the destination and waits

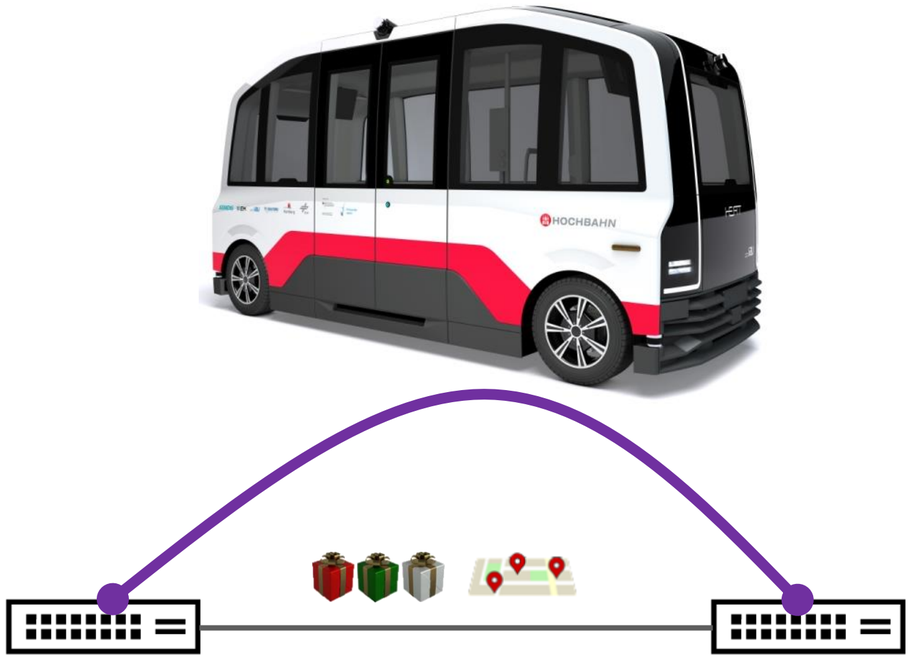
second vehicle,
travels via stops in between aka
the “spines”

Or:
One vehicle carrying both and stops in between

ALQ

ANALOGY

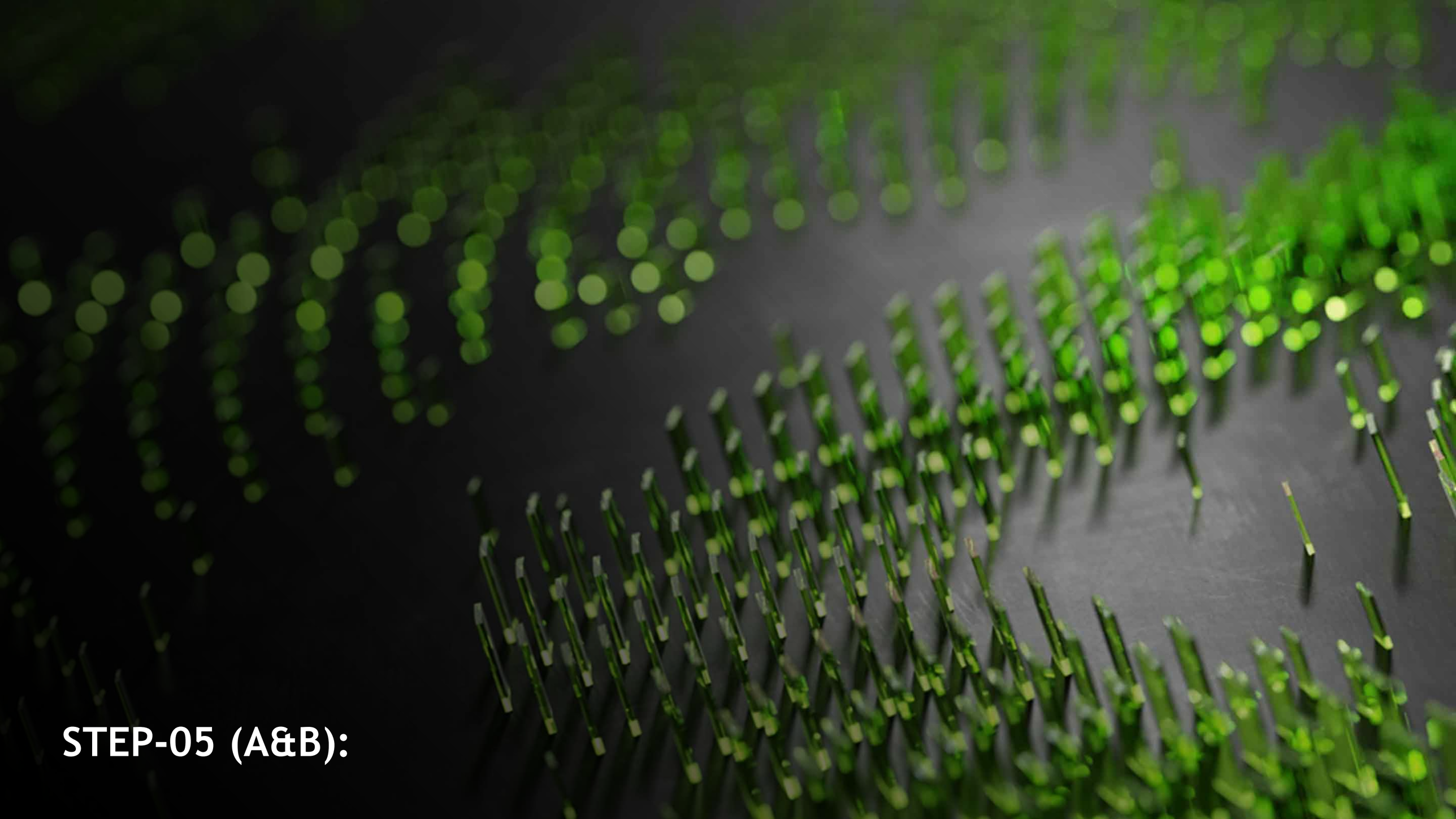
We have started “over here” with self-driving public transport options… BGP unnumbered



Connection options:
IPv6 LLA (BGP unnumbered)
IPv6
IPv4

HEAT == Hamburg Electric Autonomous Transportation

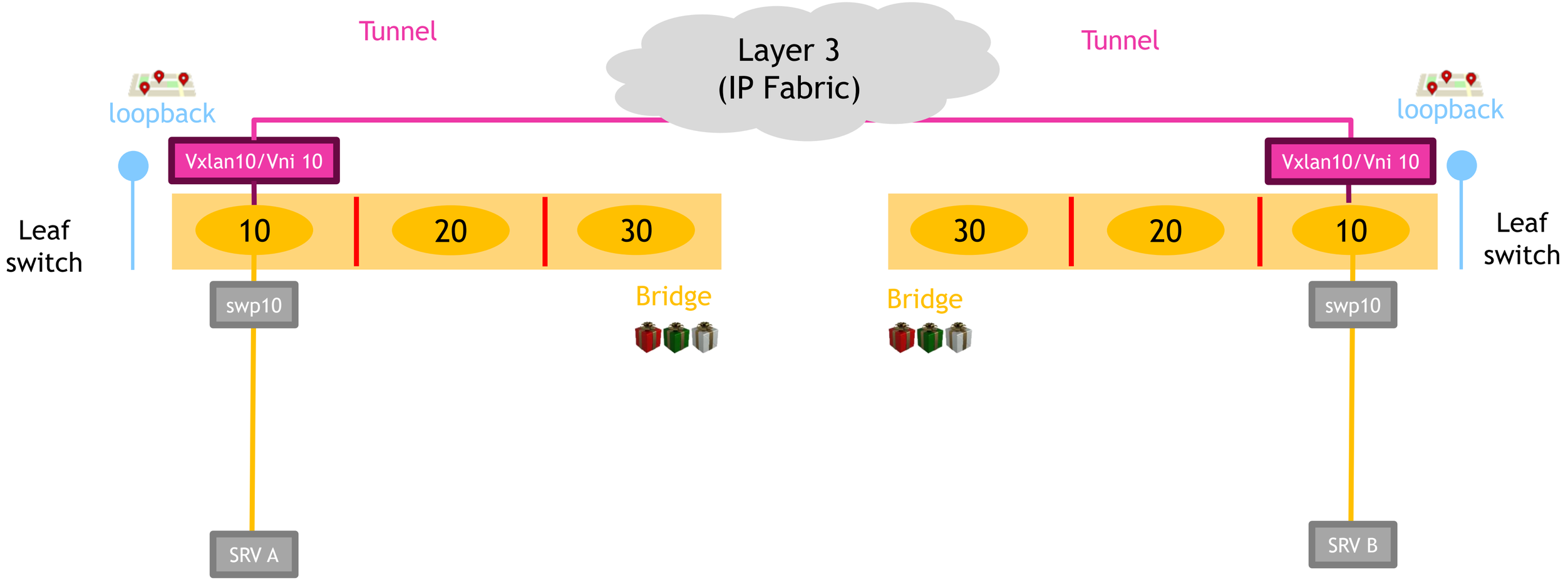
ALQ



STEP-05 (A&B):

VXLAN

The basic concept of step-05

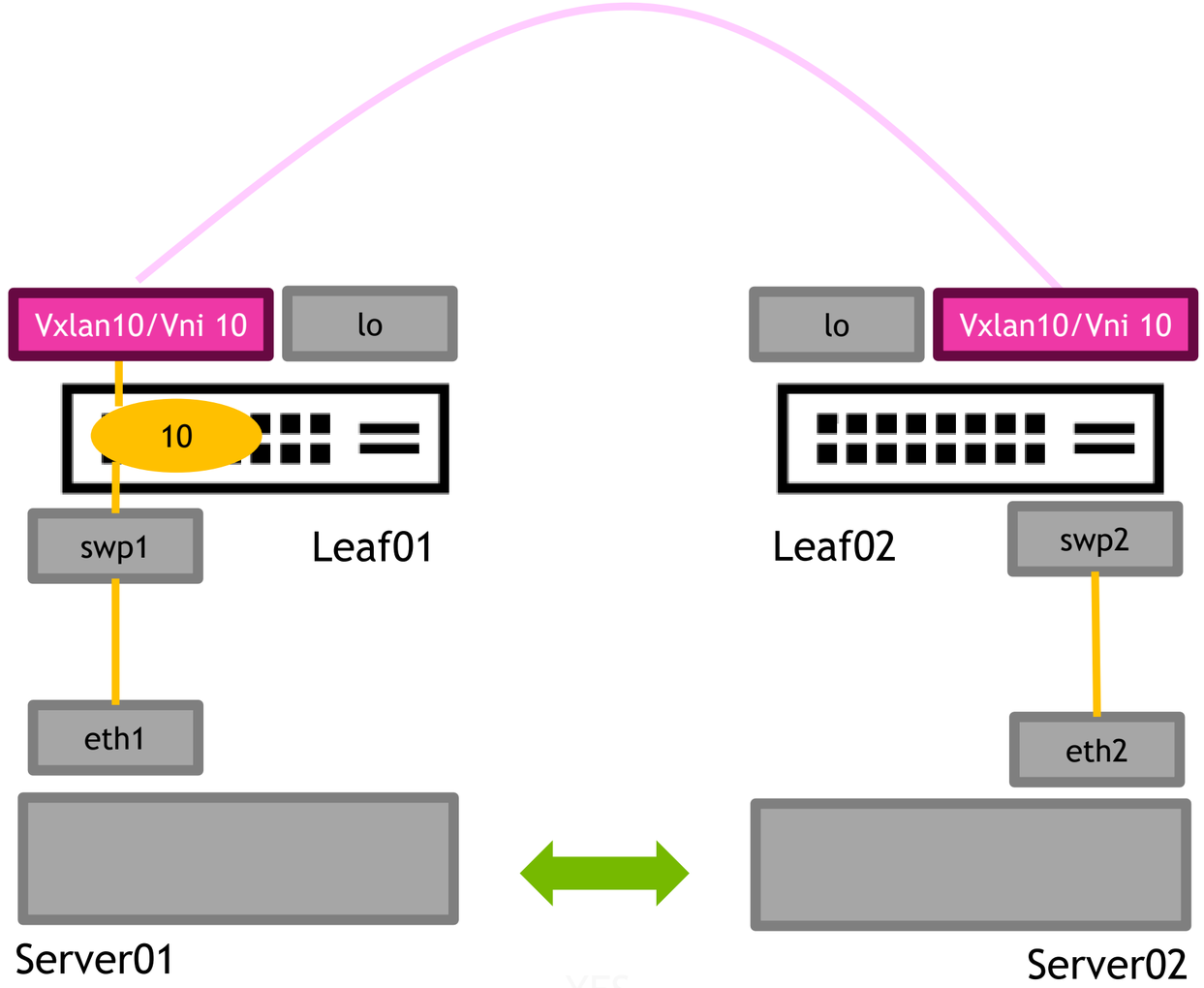


VXLAN

(static example to show an vxlan interfae)

Vxlan10/Vni 10

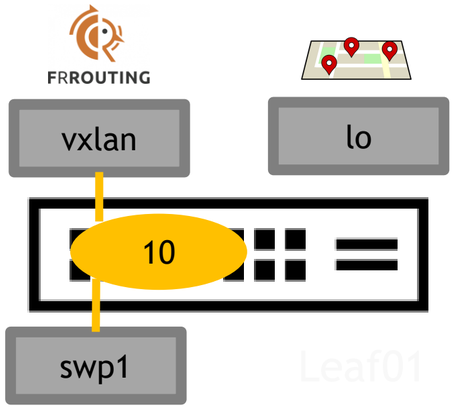
```
cumulus@leaf01:mgmt:~$ ifquery vxlan10
auto vxlan10
iface vxlan10
        bridge-access 10
        bridge-arp-nd-suppress off
        bridge-learning on
        mstpctl-bpduguard yes
        mstpctl-portbpdufilter yes
        vxlan-id 10
        vxlan-local-tunnelip 192.168.0.1
        vxlan-remoteip 192.168.0.2
```

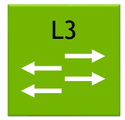


ALQ

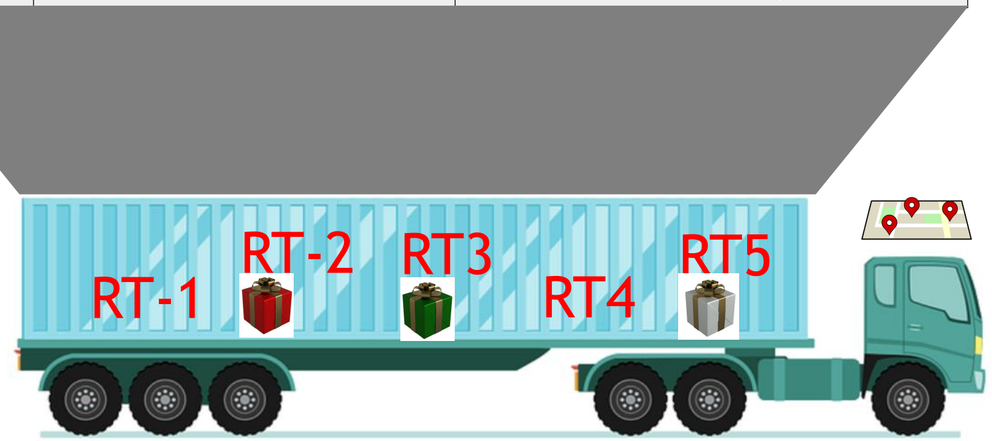
EVPN-VXLAN

Pro-actively informing all "neighbors" via BGP





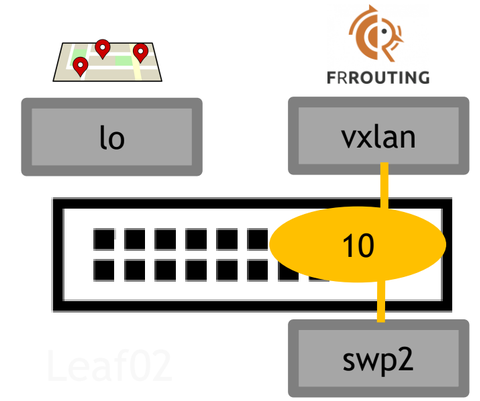
RT-2	MAC/IP Advertisement Route	Advertise MACs and/or MACIPs
RT-3	Inclusive Multicast Ethernet Tag Route	Advertise VNI membership (primarily to prune recipients of BUM traffic)
RT-5	IP Prefix Route	Advertise routes to subnet prefixes
RT-1	Ethernet AutoDiscovery (A-D) Route	For multi-homing, used to let remote VTEPs know about connectivity to an Ethernet Segment and VLANs reachable on it.
RT-4	Ethernet Segment Route	For designated forwarder (DF) election for BUM traffic handling in multi-homing scenarios.
RT-6	Selective Multicast Ethernet Tag Route	To carry IGMP multicast group membership information for a tenant using EVPN.



EVPN

EVPN MP-BGP (RFC 7432)

TCP 179



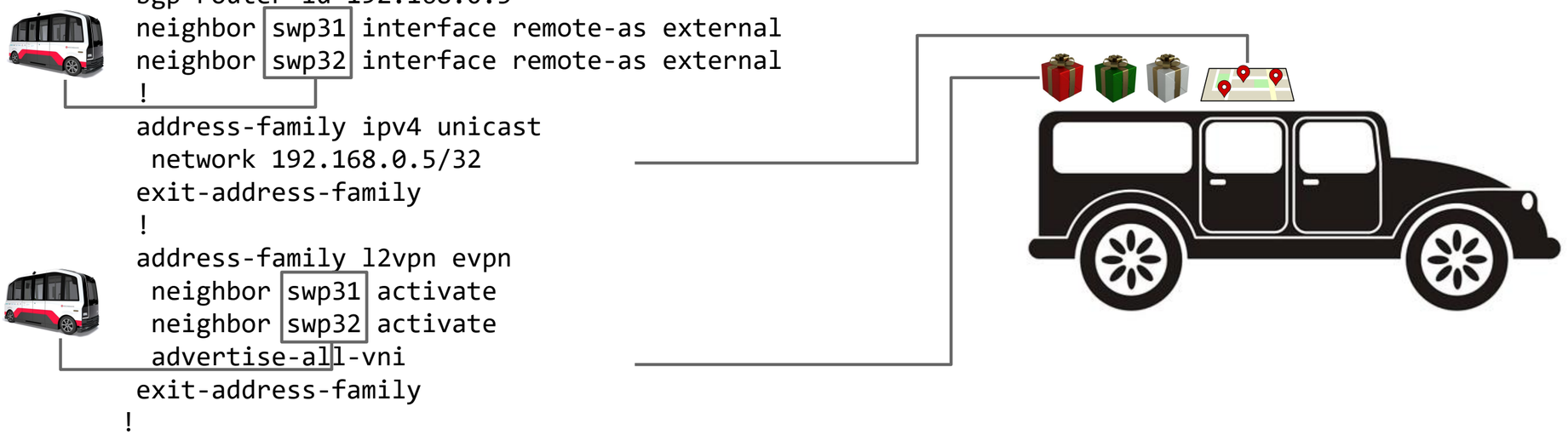
reference

ZERO-TO-EVPN

Easy Linux Networking

- Step-05: Overlay (L2-EVPN-MH)

We have already prepared BGP

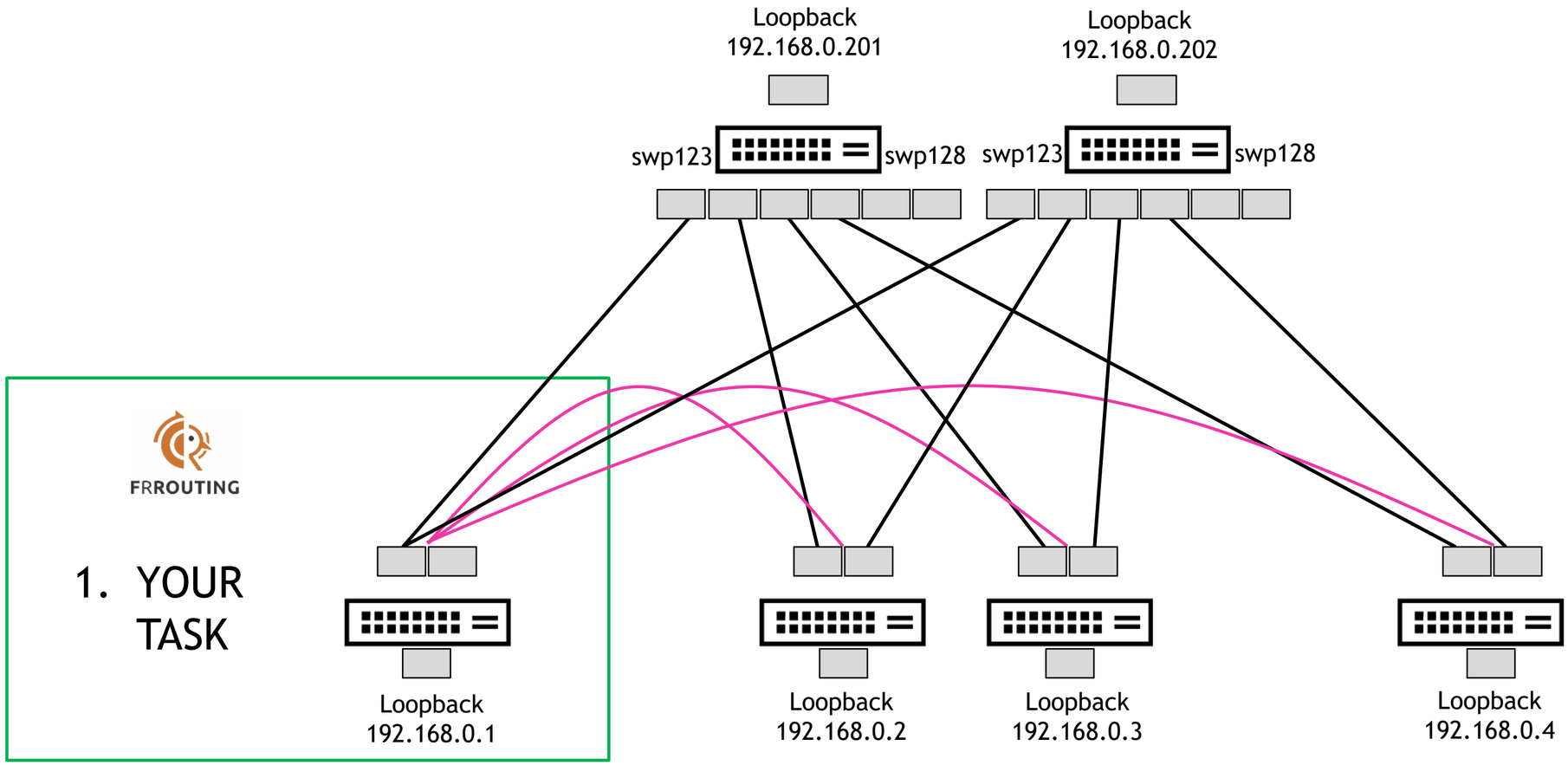


```
!
router bgp 4200000005
 bgp router-id 192.168.0.5
 neighbor swp31 interface remote-as external
 neighbor swp32 interface remote-as external
 !
 address-family ipv4 unicast
  network 192.168.0.5/32
 exit-address-family
 !
 address-family l2vpn evpn
  neighbor swp31 activate
  neighbor swp32 activate
  advertise-all-vni
 exit-address-family
!
```

ZERO-TO-EVPN

Easy Linux Networking

- Step-05: EVPN L2



L2EVPN

VXLAN Interface and BGP L2VPN EVPN Address Family

e/n/i file

```
e/n/i
<SNIP>

auto br_A
iface br_A
        bridge-ports swp1 vxlan10
        bridge-vids 10,20
        bridge-vlan-aware yes

auto vlan10
iface vlan10
        address 10.0.10.240/24
        address-virtual 00:00:5e:00:00:01 10.0.10.254/24
        vlan-id 10
        vlan-raw-device br_A

auto vxlan10
iface vxlan10
        bridge-access 10
        bridge-arp-nd-suppress on
        bridge-learning off
        mstpctl-bpduguard yes
        mstpctl-portbpdufilter yes
        vxlan-local-tunnelip 192.168.0.1
        vxlan-id 10
```

frr.conf file

```
frr.conf
<SNIP>

router bgp 4200000001
 bgp router-id 192.168.0.1
 neighbor swp31 interface remote-as external
 neighbor swp32 interface remote-as external
 !
 address-family ipv4 unicast
  network 192.168.0.1/32
 exit-address-family
 !
 address-family l2vpn evpn
  neighbor swp31 activate
  neighbor swp32 activate
  advertise-all-vni
 exit-address-family
```

TL

L2EVPN

Verification via leaf01

e/n/i file

frr.conf file

Verify on leaf01 that the remote MAC address of server11 is externally_learned

```
cumulus@leaf01:mgmt:~$ bridge fdb
44:38:39:00:00:02 dev swp1 vlan 10 master br_A permanent
44:38:39:00:00:01 dev swp1 vlan 10 master br_A
44:38:39:00:00:02 dev swp1 master br_A permanent
00:00:5e:00:00:01 dev br_A self permanent
00:00:5e:00:00:01 dev br_A vlan 10 master br_A permanent
00:00:5e:00:00:01 dev vlan10 self permanent
44:38:39:00:00:07 dev vxlan10 vlan 10 extern_learn master br_A
44:38:39:00:00:05 dev vxlan10 vlan 10 extern_learn master br_A
44:38:39:00:00:03 dev vxlan10 vlan 10 extern_learn master br_A
c6:a0:0c:28:ce:02 dev vxlan10 master br_A permanent
00:00:00:00:00:00 dev vxlan10 dst 192.168.0.2 self permanent
00:00:00:00:00:00 dev vxlan10 dst 192.168.0.3 self permanent
00:00:00:00:00:00 dev vxlan10 dst 192.168.0.4 self permanent
00:00:00:00:00:00 dev vxlan10 dst 192.168.0.5 self permanent
00:00:00:00:00:00 dev vxlan10 dst 192.168.0.6 self permanent
44:38:39:00:00:07 dev vxlan10 dst 192.168.0.4 self extern_learn
44:38:39:00:00:05 dev vxlan10 dst 192.168.0.3 self extern_learn
44:38:39:00:00:03 dev vxlan10 dst 192.168.0.2 self extern_learn
```

Verify on leaf01 the l2vpn evpn send and received number of NLRI to be larger than 0 ($ sudo vtysh)

```
leaf01# show bgp l2vpn evpn summary
BGP router identifier 192.168.0.1, local AS number 4200000001 vrf-id 0
BGP table version 0
RIB entries 11, using 2200 bytes of memory
Peers 2, using 46 KiB of memory

Neighbor        V         AS   MsgRcvd   MsgSent   TblVer  InQ OutQ  Up/Down State/PfxRcd   PfxSnt
spine01(swp31)  4 4200000201      1308      1305        0    0    0 01:03:23           10       14
spine02(swp32)  4 4200000202      1303      1300        0    0    0 01:03:18           10       14

Total number of neighbors 2
```

```
cumulus@server11:~$ ip l
<SNIP>
5: uplink: <BROADCAST,MULTICAST,MASTER,UP,LOWER_UP> mtu 9216 qdisc noqueue state UP mode DEFAULT group
default qlen 1000
    link/ether 44:38:39:00:00:05 brd ff:ff:ff:ff:ff:ff
```

```
cumulus@storage01:~$ ip l
<SNIP>
5: uplink: <BROADCAST,MULTICAST,MASTER,UP,LOWER_UP> mtu 9216 qdisc noqueue state UP mode DEFAULT group
default qlen 1000
    link/ether 44:38:39:00:00:01 brd ff:ff:ff:ff:ff:ff
```

```
cumulus@leaf01:mgmt:~$ ip l
<SNIP>
3: swp1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9216 qdisc pfifo_fast master br_A state UP mode DEFAULT
group default qlen 1000
    link/ether 44:38:39:00:00:02 brd ff:ff:ff:ff:ff:ff
<SNIP>
```

ALQ

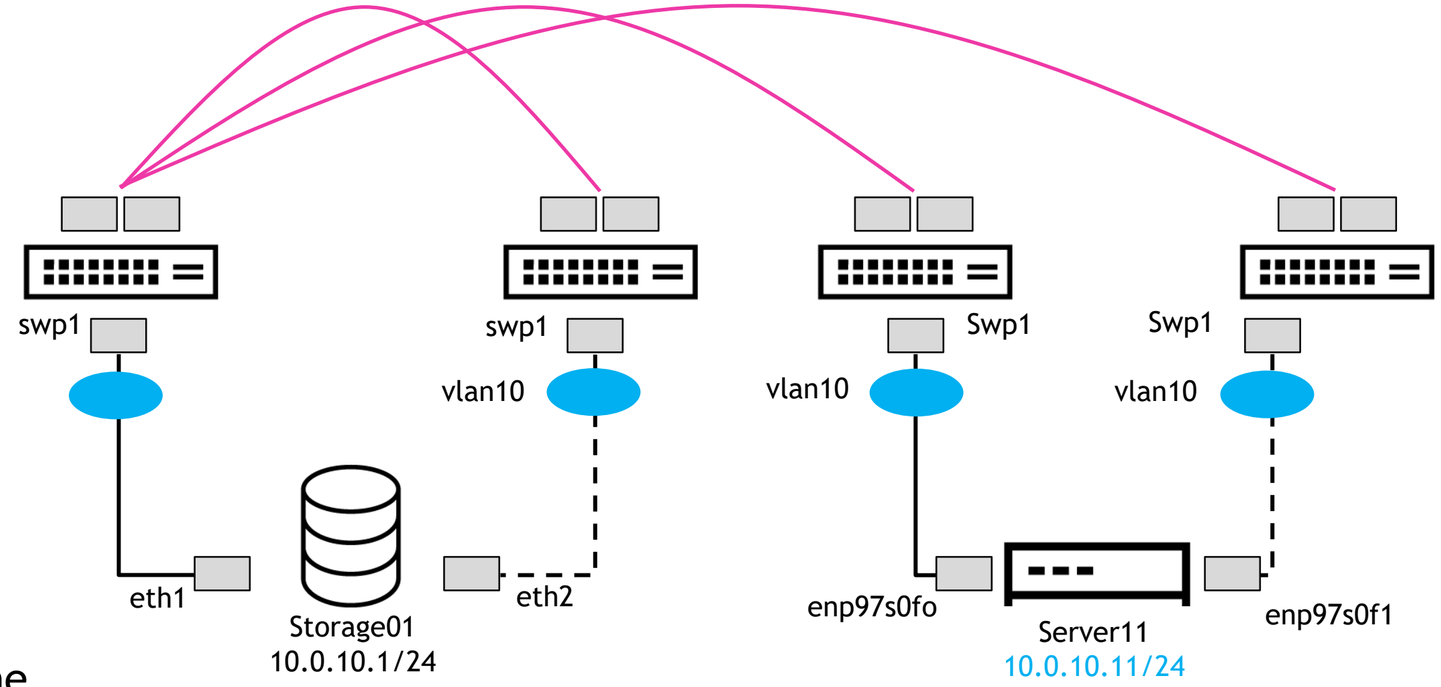
CONNECTIVITY WORKS, A GOOD RESULT

But did you notice something about the uplinks?

L2 connectivity

Verify on storage01 the reachability of server11

```
cumulus@storage01:~$ ip a
<SNIP>
5: uplink: <BROADCAST,MULTICAST,MASTER,UP,LOWER_UP> mtu 9216 /
                              qdisc noqueue state UP group default qlen 1000
    link/ether 44:38:39:00:00:01 brd ff:ff:ff:ff:ff:ff
    inet 10.0.10.1/24 scope global uplink
       valid_lft forever preferred_lft forever
    inet6 fe80::4638:39ff:fe00:1/64 scope link
       valid_lft forever preferred_lft forever
cumulus@storage01:~$ ping 10.0.10.11
PING 10.0.10.11 (10.0.10.11) 56(84) bytes of data.
64 bytes from 10.0.10.11: icmp_seq=1 ttl=64 time=2.38 ms
64 bytes from 10.0.10.11: icmp_seq=2 ttl=64 time=1.06 ms
64 bytes from 10.0.10.11: icmp_seq=3 ttl=64 time=1.19 ms
^C
--- 10.0.10.11 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 1.066/1.549/2.388/0.595 ms
```



bonds

Does the bond work
On leaf01 and storage01?
Bond-Troubleshooting hints on the following reference slides

In case of need just run the reference script

```
cumulus@oob-mgmt-server:~/ON-15$
play-step-05a-reference....sh
```

ALQ

VERIFY THE BOND

Troubleshooting (example shows the working state for reference)

```
cumulus@leaf02:mgmt:~$ sudo tcpdump -i swp1 -evv ether proto 0x8809
```

```
06:21:43.094510 44:38:39:00:00:04 (oui Unknown) > 01:80:c2:00:00:02 (oui Unknown), ethertype Slow Protocols (0x8809), length
124: LACPv1, length 110
        Actor Information TLV (0x01), length 20
          System 44:38:39:ff:ff:01 (oui Unknown), System Priority 65535, Key 9, Port 1, Port Priority 255
          State Flags [Activity, Timeout, Aggregation, Synchronization, Collecting, Distributing]
          0x0000:  ffff 4438 39ff ff01 0009 00ff 0001 3f00
          0x0010:  0000
        Partner Information TLV (0x02), length 20
          System 44:38:39:00:00:01 (oui Unknown), System Priority 65535, Key 9, Port 2, Port Priority 255
          State Flags [Activity, Timeout, Aggregation, Synchronization, Collecting, Distributing]
          0x0000:  ffff 4438 3900 0001 0009 00ff 0002 3f00
          0x0010:  0000
        Collector Information TLV (0x03), length 16
          Max Delay 0
          0x0000:  0000 0000 0000 0000 0000 0000 0000
        Terminator TLV (0x00), length 0
```

reference

VERIFY THE BOND

Troubleshooting (example shows the working state for reference)

```
cumulus@leaf02:mgmt:/proc/net/bonding$ cat bond1
Ethernet Channel Bonding Driver: v3.7.1 (April 27, 2011)

Bonding Mode: IEEE 802.3ad Dynamic link aggregation
Transmit Hash Policy: layer3+4 (1)
MII Status: up
MII Polling Interval (ms): 100
Up Delay (ms): 0
Down Delay (ms): 0

802.3ad info
LACP rate: fast
Min links: 1
Aggregator selection policy (ad_select): stable

Slave Interface: swp1
MII Status: up
Speed: 1000 Mbps
Duplex: full
Link Failure Count: 0
Permanent HW addr: 44:38:39:00:00:04
Slave queue ID: 0
Aggregator ID: 1
Actor Churn State: none
Partner Churn State: none
Actor Churned Count: 0
Partner Churned Count: 0
```

reference

VERIFY THE BOND

Troubleshooting (example shows the working state for reference)

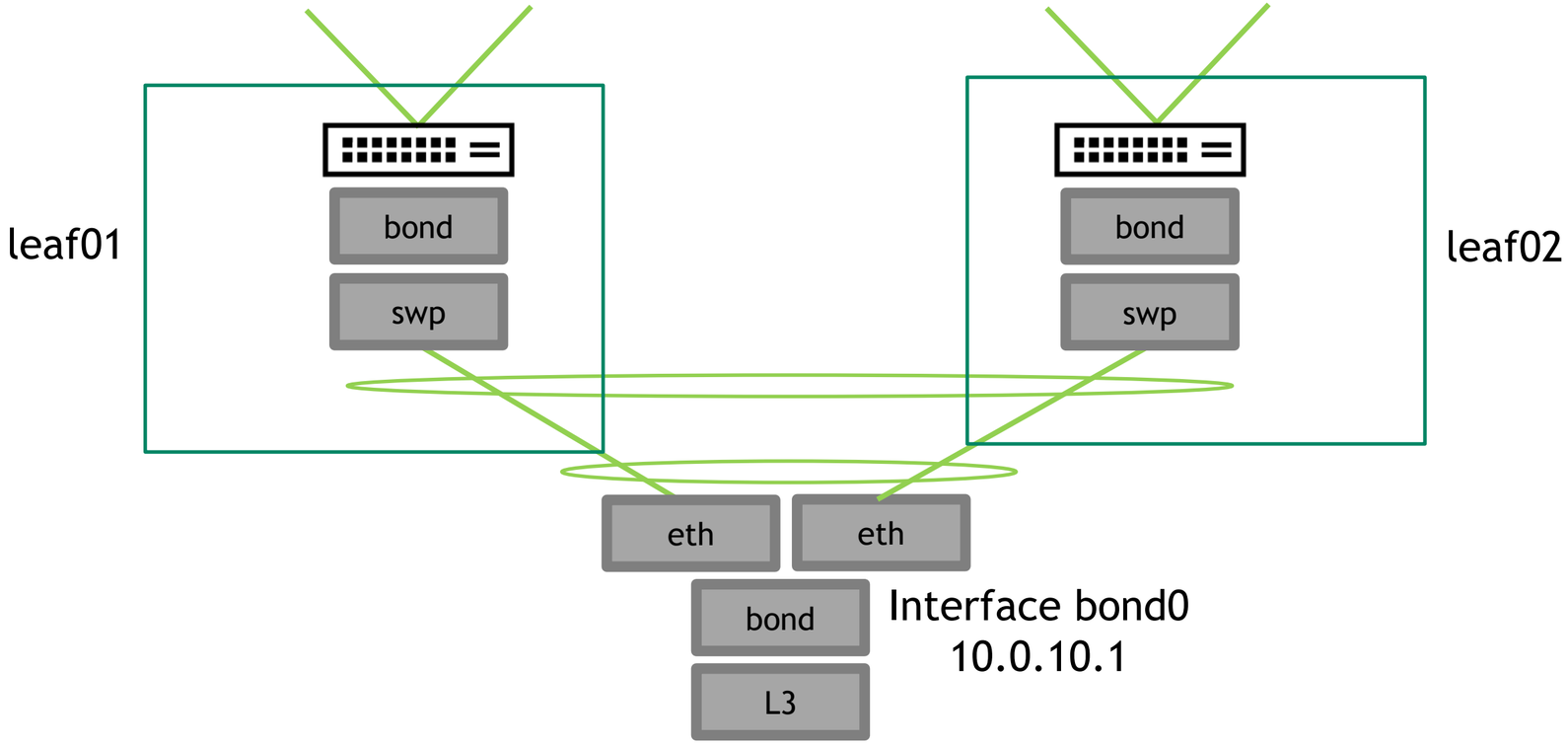
```
cumulus@leaf02:mgmt:~$ ip link xstats type bond
bond1
                     LACPDU Rx 121198
                     LACPDU Tx 121201
                     LACPDU Unknown type Rx 0
                     LACPDU Illegal Rx 0
                     Marker Rx 0
                     Marker Tx 0
                     Marker response Rx 0
                     Marker response Tx 0
                     Marker unknown type Rx 0
```

```
cumulus@leaf02:mgmt:/proc/net/bonding$ cat bond1
Ethernet Channel Bonding Driver: v3.7.1 (April 27, 2011)

Bonding Mode: IEEE 802.3ad Dynamic link aggregation
Transmit Hash Policy: layer3+4 (1)
MII Status: up
MII Polling Interval (ms): 100
Up Delay (ms): 0
Down Delay (ms): 0

802.3ad info
LACP rate: fast
Min links: 1
Aggregator selection policy (ad_select): stable

Slave Interface: swp1
MII Status: up
Speed: 1000 Mbps
Duplex: full
Link Failure Count: 0
Permanent HW addr: 44:38:39:00:00:04
Slave queue ID: 0
Aggregator ID: 1
Actor Churn State: none
Partner Churn State: none
Actor Churned Count: 0
Partner Churned Count: 0
```

reference

ZERO-TO-EVPN

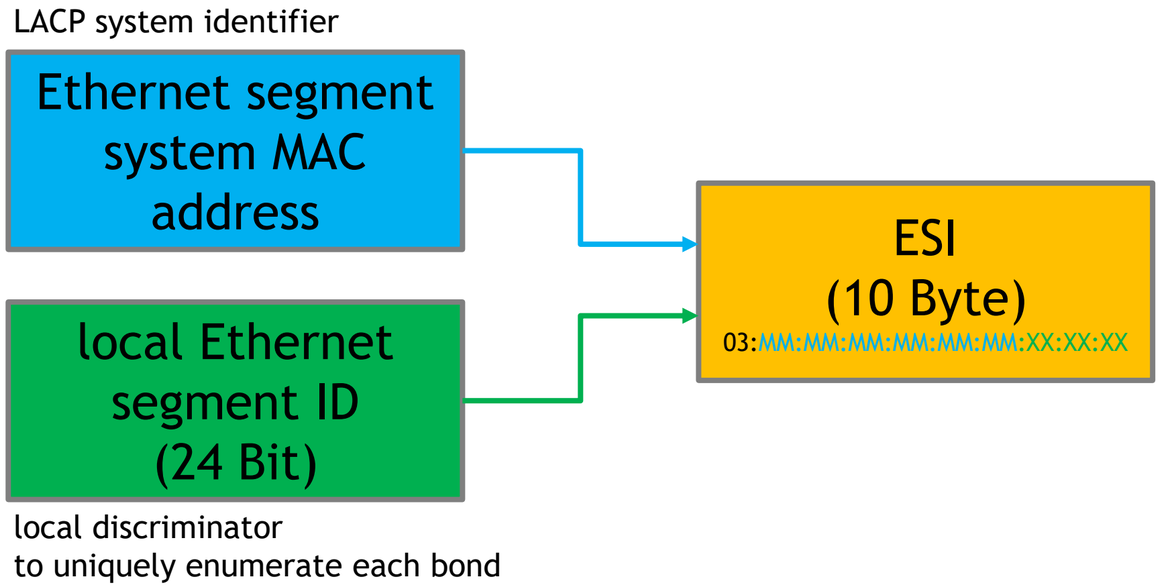
The basic concept of step-05-MH



EVPN-MH

Multi-Homing

- components



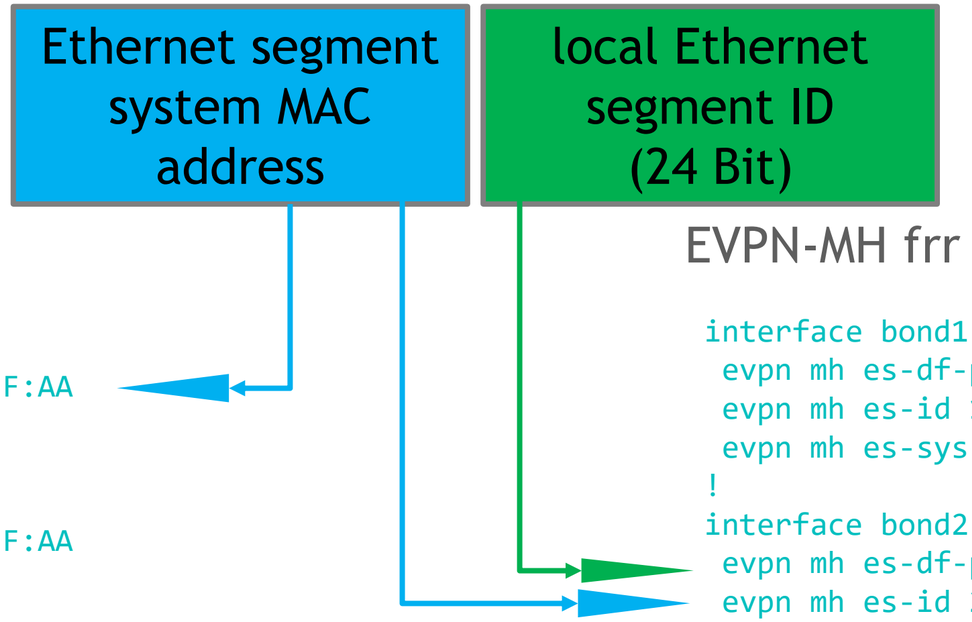

```
#evpn.multihoming.enable=TRUE
#evpn.multihoming.shared_l2_groups=FALSE
#evpn.multihoming.shared_l3_groups=FALSE
```

ALQ

EVPN-MH

Multi-Homing

- EVPN-MH /e/n/i



ESI (10 Byte)

03:MM:MM:MM:MM:MM:MM:XX:XX:XX

03:44:38:39:ff:ff:01:00:00:01

```
interface bond1
  bond-slaves swp1
  es-sys-mac 44:38:39:BE:EF:AA

interface bond2
  bond-slaves swp2
  es-sys-mac 44:38:39:BE:EF:AA
```

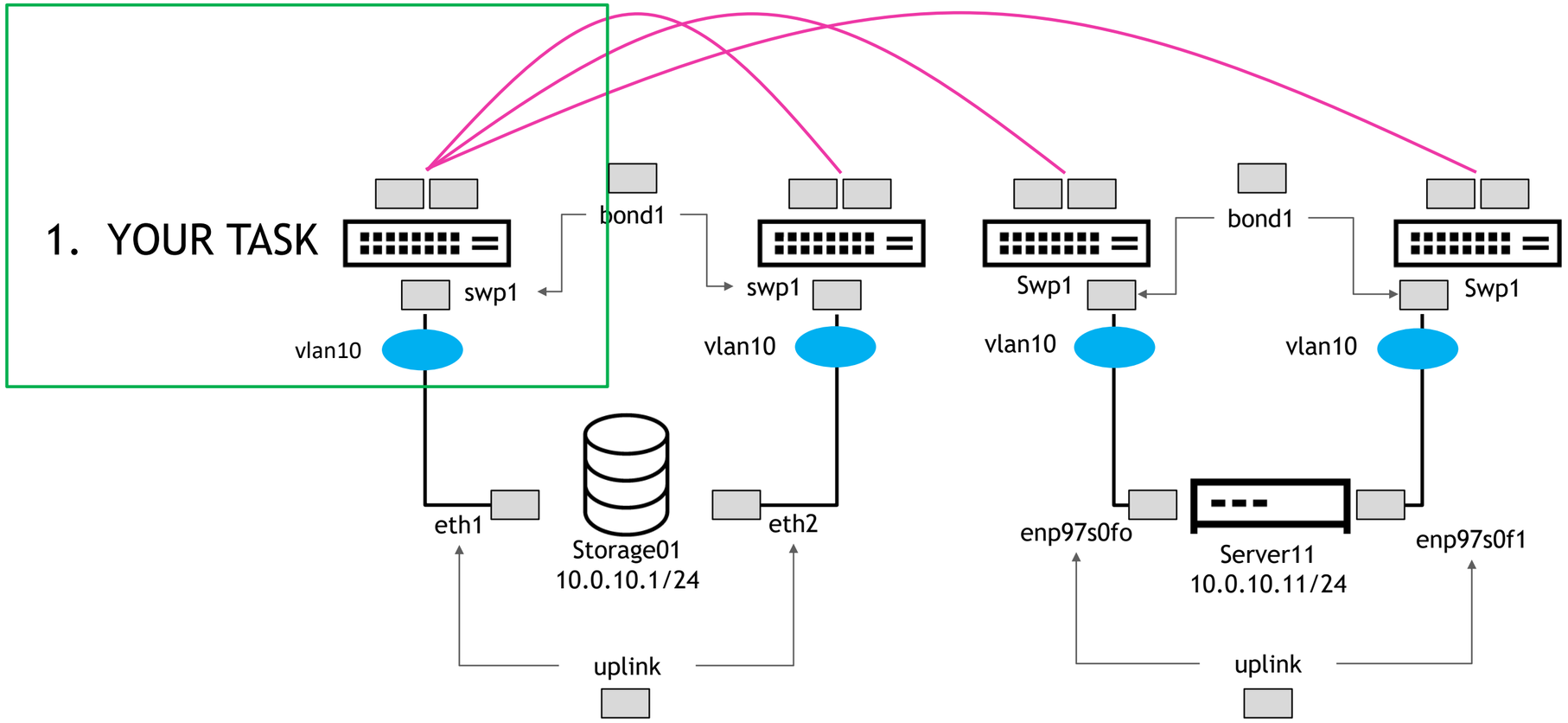
EVPN-MH frr

```
interface bond1
 evpn mh es-df-pref 50000
 evpn mh es-id 1
 evpn mh es-sys-mac 44:38:39:BE:EF:AA
!
interface bond2
 evpn mh es-df-pref 50000
 evpn mh es-id 2
 evpn mh es-sys-mac 44:38:39:BE:EF:AA

!
interface swp1
 evpn mh uplink
!
interface swp2
 evpn mh uplink
!
interface swp3
 evpn mh uplink
!
interface swp4
 evpn mh uplink
!
```

ALQ

STEP-05-L2-MH

EVPN-MH playbook includes both a e/n/I and an frr file



LAB AND/OR BREAK TIME

Step-05a: 10 min (EVPN-Layer2)
Step-05b: 15 min (EVPN-MH)

VERIFICATION & SUMMARY

& next steps

EVPN MH

Verify on leaf01 the EVPN-MH setup

Bridge FDB

```
leaf01# show evpn l2-nh
VTEP          NH id     #ES
192.168.0.4     268435461  1
192.168.0.2     268435458  1
192.168.0.3     268435460  1

leaf01# show evpn es
Type: B bypass, L local, R remote, N non-DF
ESI                     Type ES-IF          VTEPs
03:44:38:39:ff:ff:01:00:00:01  LR   bond1            192.168.0.2
03:44:38:39:ff:ff:11:00:00:01  R    -                192.168.0.3,192.168.0.4
```

```
cumulus@leaf01:mgmt:~$ bridge fdb
00:00:5e:00:00:01 dev br_A self permanent
44:38:39:00:00:02 dev br_A vlan 10 master br_A permanent
00:00:5e:00:00:01 dev br_A vlan 10 master br_A permanent
00:00:5e:00:00:01 dev vlan10 self permanent
44:38:39:00:00:07 dev vxlan10 vlan 10 extern_learn master br_A
44:38:39:00:00:05 dev vxlan10 vlan 10 extern_learn master br_A
44:38:39:00:00:08 dev vxlan10 vlan 10 extern_learn master br_A
44:38:39:00:00:06 dev vxlan10 vlan 10 extern_learn master br_A
44:38:39:00:00:04 dev vxlan10 vlan 10 extern_learn master br_A
4e:13:2a:8f:21:73 dev vxlan10 vlan 10 master br_A permanent
44:38:39:00:00:16 dev vxlan10 vlan 10 extern_learn master br_A
44:38:39:00:00:1a dev vxlan10 vlan 10 extern_learn master br_A
4e:13:2a:8f:21:73 dev vxlan10 master br_A permanent
00:00:00:00:00:00 dev vxlan10 dst 192.168.0.2 self permanent
00:00:00:00:00:00 dev vxlan10 dst 192.168.0.3 self permanent
00:00:00:00:00:00 dev vxlan10 dst 192.168.0.4 self permanent
00:00:00:00:00:00 dev vxlan10 dst 192.168.0.5 self permanent
00:00:00:00:00:00 dev vxlan10 dst 192.168.0.6 self permanent
44:38:39:00:00:08 dev vxlan10 dst 192.168.0.4 self extern_learn
44:38:39:00:00:07 dev vxlan10 nhid 536870915 self extern_learn
44:38:39:00:00:06 dev vxlan10 dst 192.168.0.3 self extern_learn
44:38:39:00:00:05 dev vxlan10 nhid 536870915 self extern_learn
44:38:39:00:00:04 dev vxlan10 dst 192.168.0.2 self extern_learn
44:38:39:00:00:1a dev vxlan10 dst 192.168.0.6 self extern_learn
44:38:39:00:00:16 dev vxlan10 dst 192.168.0.5 self extern_learn
44:38:39:00:00:01 dev bond1 vlan 10 master br_A static
44:38:39:00:00:03 dev bond1 vlan 10 master br_A static
44:38:39:00:00:02 dev bond1 master br_A permanent
```

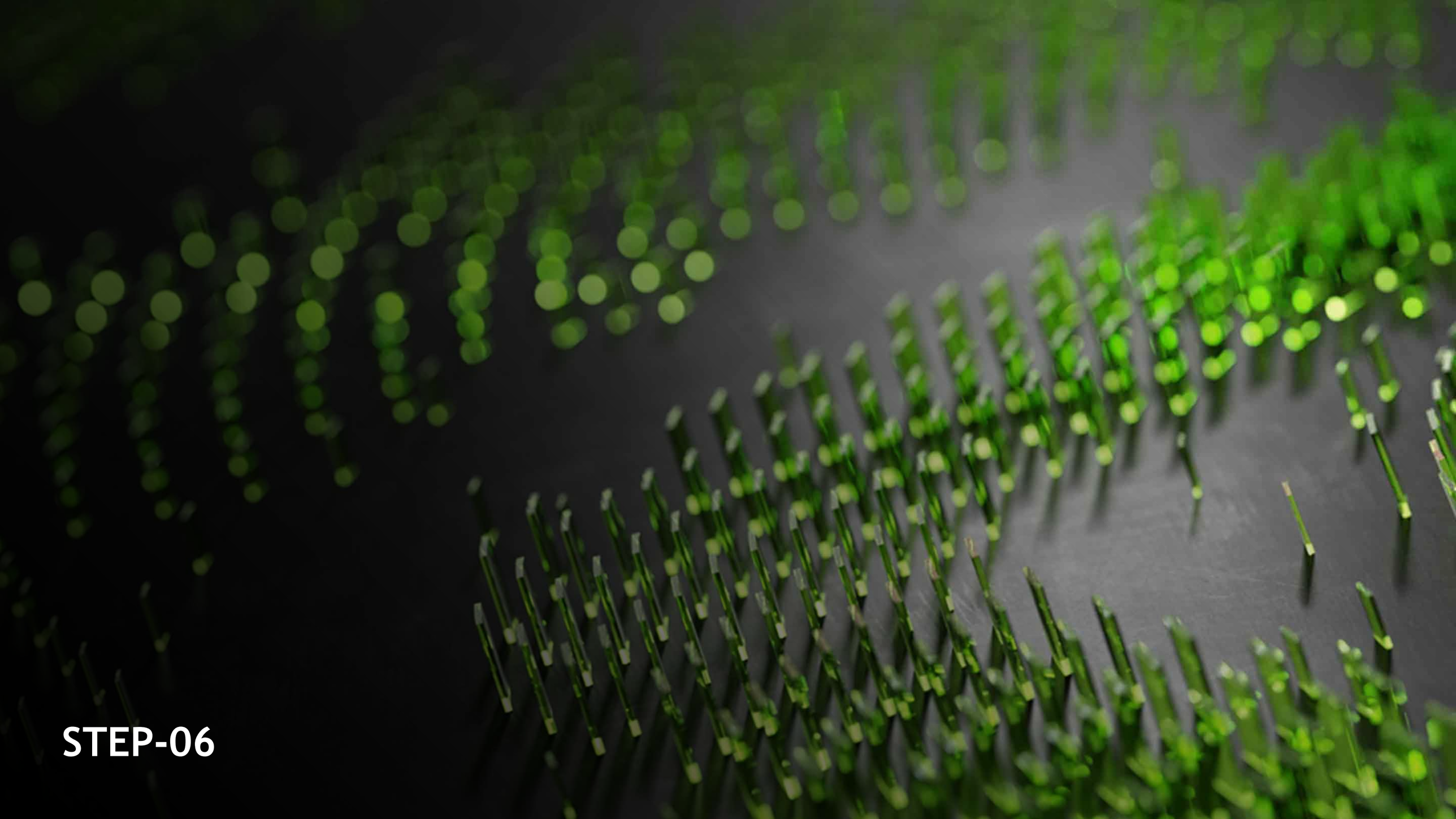
- We have created Layer 2 connectivity (Storage01 to Server11) via the Layer 3 Infrastructure
- Compute and storage nodes are dual-homed via Layer 2

- Next step:
- We need layer 3 connectivity for the first Tenant (Tenant_A) to enable Storage01 being accessible from server13

In case of need just run the reference script

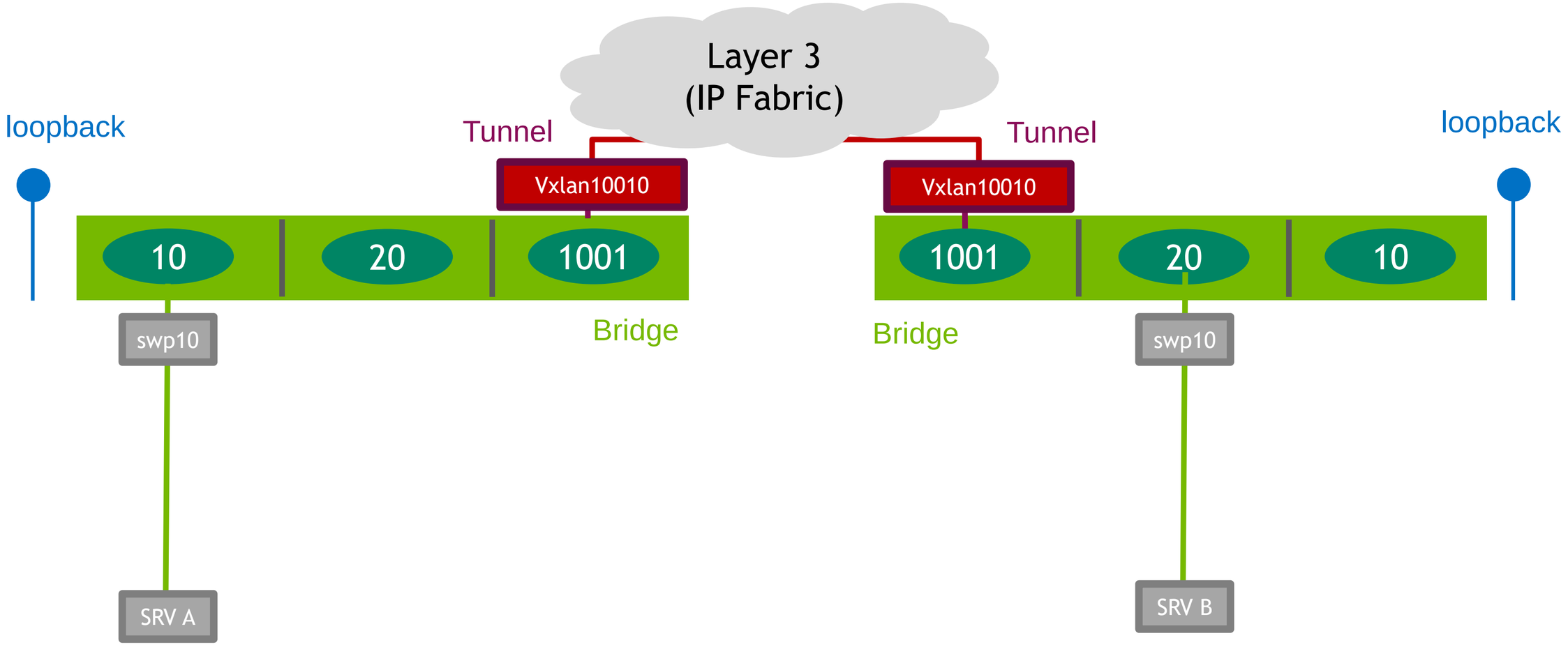
```
cumulus@oob-mgmt-server:~/ON-15$
Play-step-05b-reference....sh
```

ALQ


STEP-06

ZERO-TO-EVPN

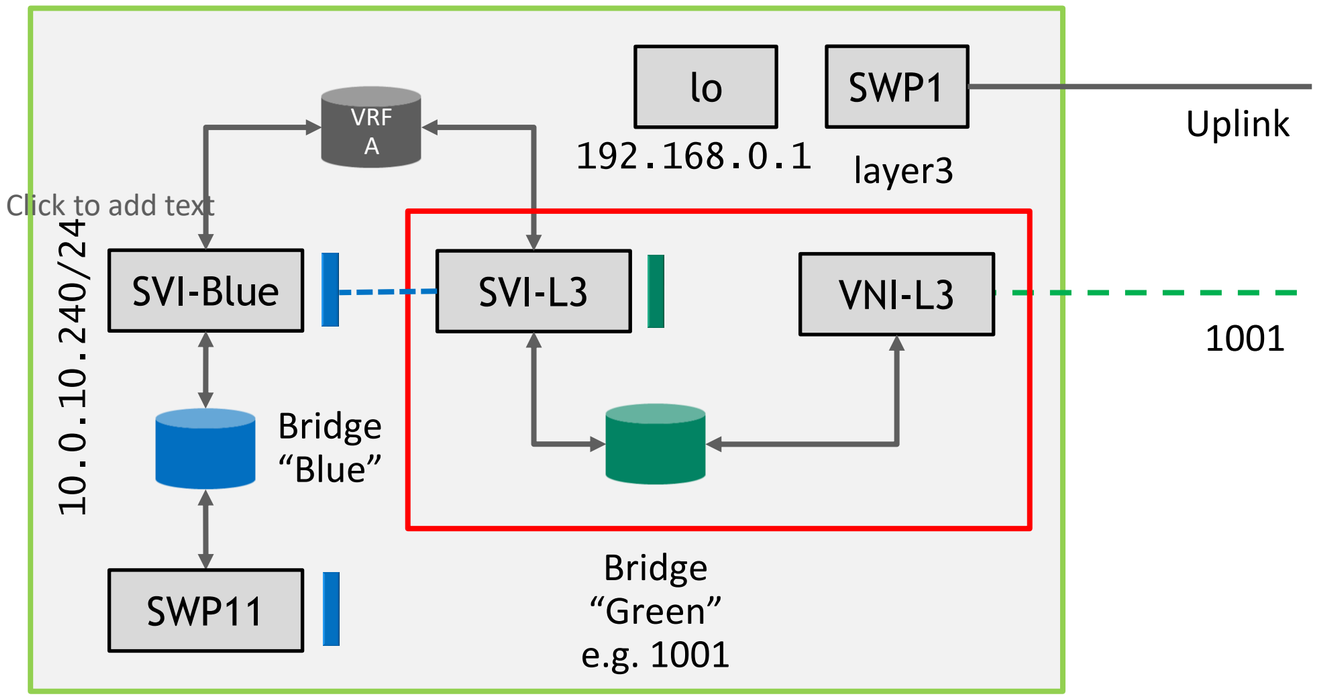
The basic concept of step-06



DISTRIBUTED SYMMETRICAL ROUTING

- Highest Scaling implementation
- Requirement for EVPN-MH

```
auto VNI-L3
iface VNI-L3
        bridge-access 1001
        mstpctl-bpduguard yes
        mstpctl-portbpdufilter yes
        vxlan-id 1001
        vxlan-local-tunnelip 10.0.0.1

auto bridge
iface bridge
        bridge-ports swp11 VNI-L3
        bridge-vids 10 1001
        bridge-vlan-aware yes

auto SVI-L3
iface SVI-L3
        vlan-id 1001
        vlan-raw-device bridge
        vrf VRFA

auto SVI-Blue
iface SVI-BLue
        address 172.16.10.252/24
        vlan-id 10
        vlan-raw-device bridge
        vrf VRFA
```

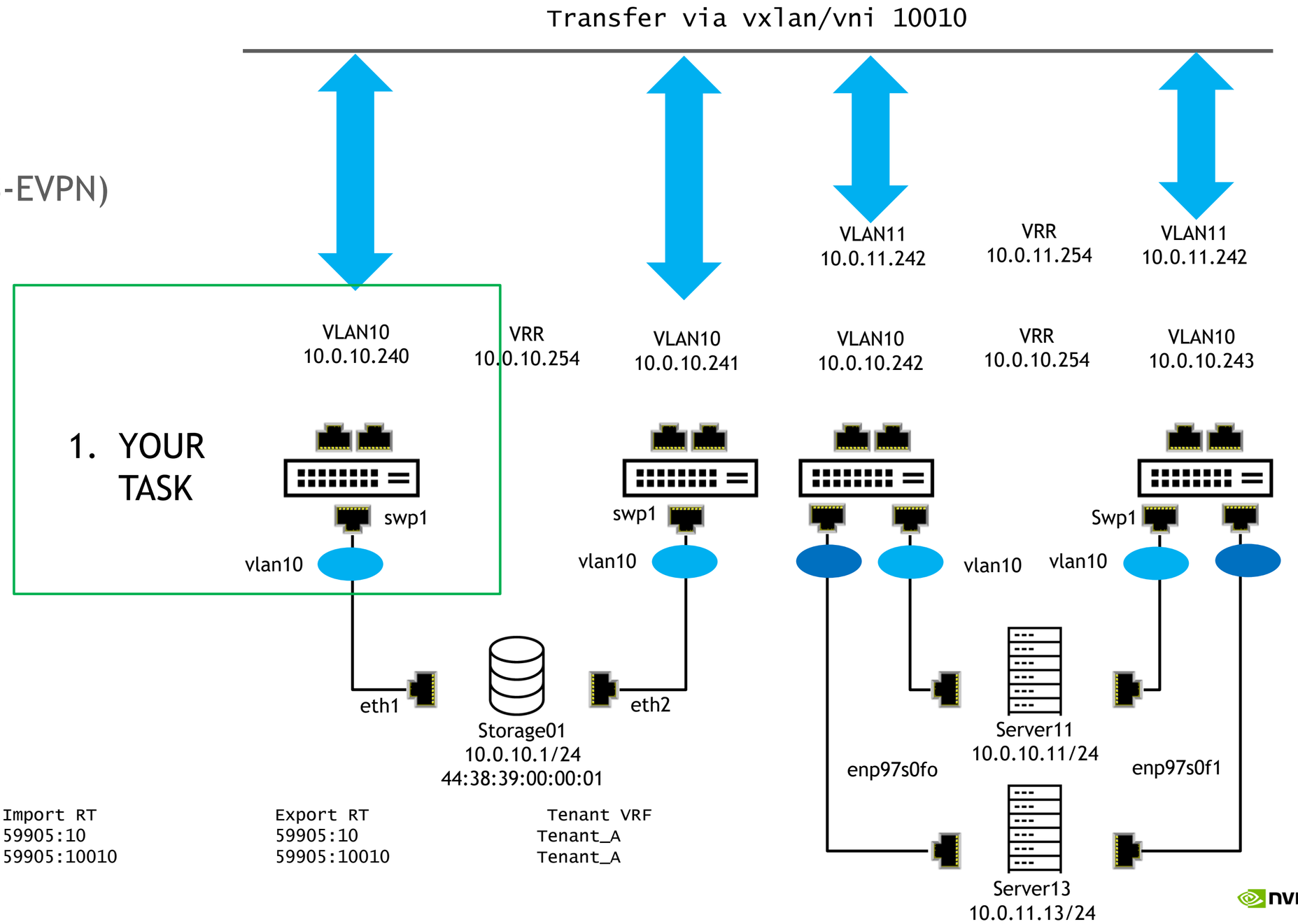


ALQ

ZERO-TO-EVPN

Easy Linux Networking

Step-06: Overlay (L3-EVPN)




```
leaf01# sh bgp l2vpn evpn vni
Advertise Gateway Macip: Disabled
Advertise SVI Macip: Disabled
Advertise All VNI flag: Enabled
BUM flooding: Head-end replication
Number of L2 VNIs: 1
Number of L3 VNIs: 1
Flags: * - Kernel
  VNI        Type RD                    Import RT           Export RT           Tenant VRF
* 10         L2   192.168.0.1:2         59905:10            59905:10            Tenant_A
* 10010      L3   10.0.10.254:4         59905:10010         59905:10010         Tenant_A
```

TL

LAB AND/OR BREAK TIME

Step-06a: 10 min

ZERO-TO-EVPN

Easy Linux Networking

Step-06: verification

VNI

Verify on leaf01 the presence of two VNI's one for L2(10) and one for L3(10010)

```
leaf01# sh bgp l2vpn evpn vni
Advertise Gateway Macip: Disabled
Advertise SVI Macip: Disabled
Advertise All VNI flag: Enabled
BUM flooding: Head-end replication
Number of L2 VNIs: 1
Number of L3 VNIs: 1
Flags: * - Kernel
  VNI        Type RD                    Import RT                 Export RT                 Tenant VRF
* 10         L2   192.168.0.1:2         59905:10                  59905:10                  Tenant_A
* 10010      L3   10.0.10.254:4         59905:10010               59905:10010               Tenant_A
```

L2 connectivity

Verify on storage01 the reachability of server13

In case of need just run the reference script

```
cumulus@oob-mgmt-server:~/ON-15$
play-step-06a-reference....sh
```

ZERO-TO-EVPN

Easy Linux Networking (optional, demo is prepared for you but based on time remaining)

Step-06b: Faster, more efficient just 20 lines just run play-step-06b-reference...

20 lines (18 sloc) | 506 Bytes

```
1   ---
2   - hosts: leaf
3     name: TOR
4     become: yes
5     gather_facts: no
6     tasks:
7       - name: copy eni
8         copy:
9           src: /home/cumulus/ON-15/step-06/{{ inventory_hostname }}-if
10          dest: /etc/network/interfaces
11      - name: activate changes on leaf01
12        shell: /sbin/ifreload -a
13
14      - name: copy frr
15        copy:
16          src: /home/cumulus/ON-15/step-06/{{ inventory_hostname }}-frr
17          dest: /etc/frr/frr.conf
18      - name: reload frr
19        ansible.builtin.shell: systemctl reload frr
20
```

```
<SNIP>
[leaf]
leaf01 ansible_host=192.168.200.9
leaf02 ansible_host=192.168.200.10
leaf03 ansible_host=192.168.200.11
leaf04 ansible_host=192.168.200.12
leaf05 ansible_host=192.168.200.13
leaf06 ansible_host=192.168.200.14

[task-leaf]
leaf02 ansible_host=192.168.200.10
leaf03 ansible_host=192.168.200.11
leaf04 ansible_host=192.168.200.12
leaf05 ansible_host=192.168.200.13
leaf06 ansible_host=192.168.200.14

[spine]
spine01 ansible_host=192.168.200.15
spine02 ansible_host=192.168.200.16

[switche:children]
leaf
spine

[task-switches:children]
task-leaf
Spine
<SNIP>
```

TL

ZERO-TO-EVPN

Easy Linux Networking, optionally if you are "fast/advanced" or we have time...

Step-06c: templates — More efficiency by using Jinja

28 lines (23 sloc) | 829 Bytes

```
---
- hosts: leaf01
  name: task-step
  become: yes
  gather_facts: no

  tasks:
    - name: render file via template
      template:
        src: /home/cumulus/ON-15/step-06c/templates/leaf01-if.j2
        dest: /etc/network/interfaces
    - name: kick it
      ansible.builtin.shell: /sbin/ifreload -a

    - name: copy frr
      copy:
        src: /home/cumulus/ON-15/step-06c/leaf01-frr
        dest: /etc/frr/frr.conf
    - name: reload frr
      ansible.builtin.shell: systemctl reload frr

# ************************************************************
# *****            Don't change anything below            *****
# ************************************************************
```

83 lines (69 sloc) | 1.45 KB

```
{% set ip_lo = "7.7.7.7/32" %}

auto lo
iface lo inet loopback
    address {{ ip_lo }}

auto mgmt
iface mgmt
        address 127.0.0.1/8
        address ::1/128
        vrf-table auto

auto Tenant_A
iface Tenant_A
        vrf-table auto

auto Tenant_B
iface Tenant_B
        vrf-table auto
```

LAB 6C (OPTIONAL)

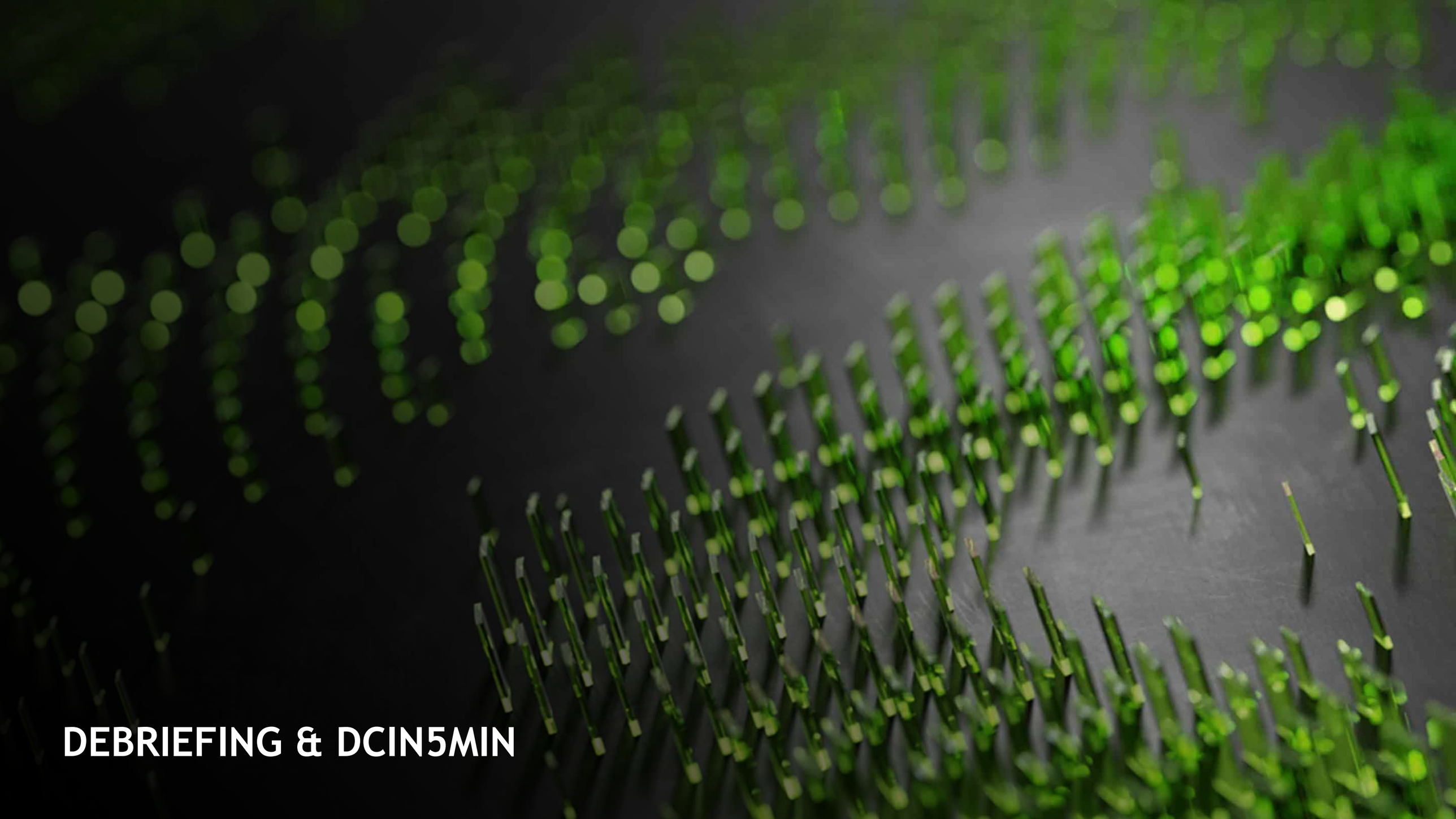
Step-06c: (self-paced)

ALQ

LAB 07 (OPTIONAL)

Step-07: (self-paced)

ALQ


DEBRIEFING & DCIN5MIN

DEBRIEFING

The building blocks

- Network devices and compute nodes can be configured and operated in the same way
- The compute orchestration tools can be expanded to cover the networking portion of the data center
- Modern enterprise data center often use a layer 3 based leaf-spine architecture with overlays
- Overlays can be dynamically setup via a vendor independent and interoperable distributed control-plane (BGP)
- Configuration of networking devices can be based on

  - NVUE (NVIDIA User Experience)
    - CLI commands
    - YAML files

  - Linux “Flat Files”

  - IP(route2) + FileSystem+Ethtool + Routing Engine instructions

ALQ+TL

DEBRIEFING

The building blocks (we covered today a selection of an even wider range…)

- Option-01: Sending NVUE commands

- Option-02: Sending a NVUE YAML file to replace or patch

- Option-03: Sending Linux Files like /e/n/I, frr.conf via individual tasks and files

- Option-04: like Option-03 but looping within one task

- Option-05: like Option-04 but rendering individual files out of one template (J2)

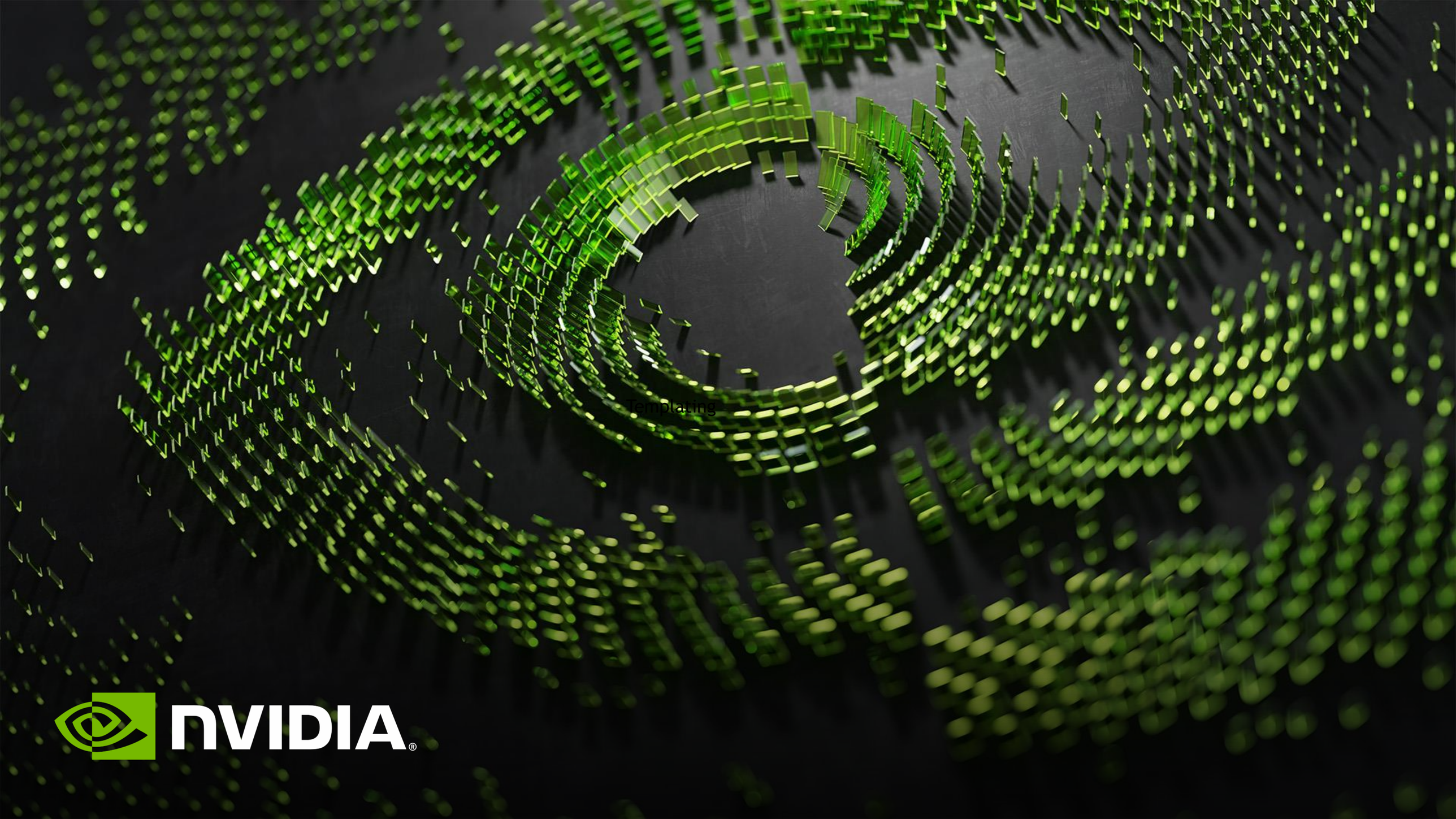
- Option-06: working with native Linux instructions (most likely for troubleshooting and corner cases)

REBUILD NODES AND CONFIGURE

air.nvidia.com simulation (optional)

- Step-01:
  - select the leaf or spine node(s) to destroy/rebuild

- Step-02:
  - run the ansible-playbooks ??? for this/those node(s)
  - Ansible command to run the play for the selected nodes…


Templating
NVIDIA

PREPARATION FOR TODAY'S TASKS OVERVIEW (BACKUP-SLIDES)

ORCHESTRATION

Step-02

IP(ROUTE2)

Simple play-books using a shell-script, e.g.:

11 lines (11 sloc) | 282 Bytes

```
1  ---
2  - hosts: hypervisor01
3    name: create vSwitch1 and vSwitch2
4    become: yes
5    gather_facts: false
6    tasks:
7      - copy:
8          src: /home/cumulus/ON-15/step-02/prepare_hypervisor/v
9          dest: /home/cumulus/vswitches.sh
10         mode: u+xrw
11     - shell: ./vswitches.sh
```

```
20     # fix until bridge binding is fixed
21     #- name:
22     #  shell: ip link {{ item }}
23     #  with_items:
24     #  - add link uplink name uplink.10 type vlan id 10
25     #  - set up dev uplink.10
26     #  shell: ip addr add 10.0.10.12/24 dev uplink.10
27     #
28     #  shell: ip link {{ item }}
29     #  with_items:
30     #  - add link uplink name uplink.20 type vlan id 20
31     #  - set up dev uplink.20
32     #  shell: ip addr add 10.0.20.12/24 dev uplink.20
```

1. CREATE IF
2. ENABLE IF
3. CONFIGURE IF

ORCHESTRATION

Step-03

Simple play-books using NVUE commands , e.g.:

1. CONFIGURE IF
2. CONFIGURE BRIDGE
3. SET HOSTNAME
4. ACTIVATE PENDING CHANGES

1. GET BRIDGE IF AND FDB
2. SHOW OUTPUT

26 lines (24 sloc) | 733 Bytes

```
1  ---
2  - hosts: leaf01
3    name: create bridge, set loopback and enable switch ports
4    become: yes
5    gather_facts: no
6    tasks:
7      - name: nvue set items
8        shell: nv set {{ item }}
9        with_items:
10       - interface lo ip address 192.168.0.1/32
11       - interface swp1,31-32
12       - bridge domain br_A vlan 10,11
13       - interface swp1 bridge domain br_A access 10
14       - platform hostname value leaf01
15
16     - name: activate staging buffer
17       shell: nv config apply -y
18     - name: iproute2 bridge interface list
19       shell: bridge link
20       register: br
21     - debug: msg={{ br.stdout }}
22     - name: iproute2 bridge forwarding database
23       shell: bridge fdb
24       register: fdb
25     - debug: msg={{ fdb.stdout }}
26
```

ORCHESTRATION

Step-03

- Alternative to individual commands we could create or render a YAML file
- Copy the file to the target(s).
- And patch, replace or apply it.

1. CREATE/USE STARTUP.YAML

```
- set:
    bridge:
      domain:
        br_A:
          vlan:
            '10': {}
            '11': {}
    platform:
      hostname:
        value: leaf01
    interface:
      lo:
        ip:
          address:
            192.168.0.1/32: {}
        type: loopback
      swp1:
        type: swp
        bridge:
          domain:
            br_A:
              access: 10
      swp31:
        type: swp
      swp32:
        type: swp
```

ORCHESTRATION

Step-04

- Linux Files, it simply works.
- Same approach for compute, storage and network.

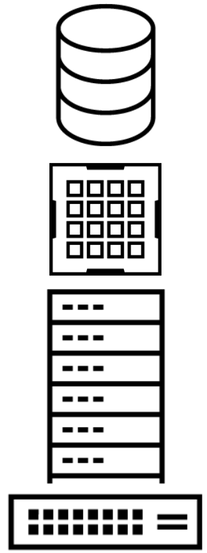
`/etc/network/interfaces`

`/etc/frr/frr.conf`

`/etc/frr/daemons`

standard modules

copy



and some more files if you like/need...

ORCHESTRATION

Step-05: EVPN-MH (Layer2)

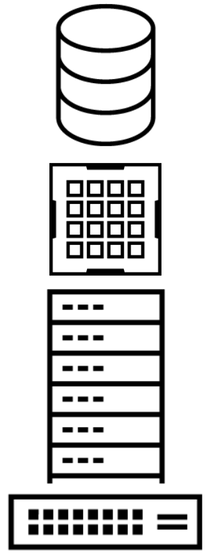
- We continue to use Linux Files.

/etc/network/interfaces

/etc/frr/frr.conf

standard modules

copy



ORCHESTRATION

Step-06: EVPN-MH (Layer3)

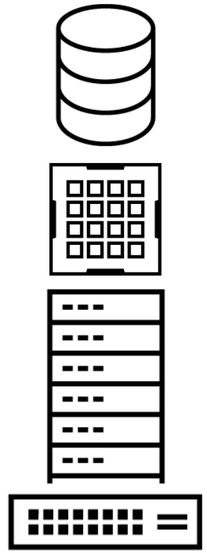
- Linux Files, it simply works.

`/etc/network/interfaces`

`/etc/frr/frr.conf`

standard modules

copy



ORCHESTRATION

Step-06b: EVPN-MH (Layer3)

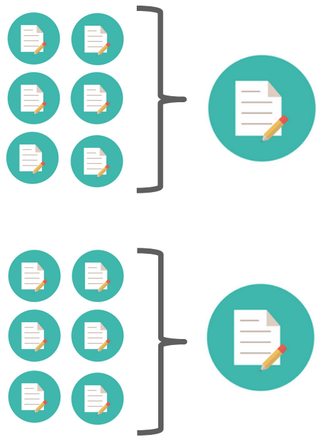
- Linux Files, it simply works.

File: hosts

```
7   [leaf]
8   leaf01 ansible_host=192.168.200.9
9   leaf02 ansible_host=192.168.200.10
10  leaf03 ansible_host=192.168.200.11
11  leaf04 ansible_host=192.168.200.12
12  leaf05 ansible_host=192.168.200.13
13  leaf06 ansible_host=192.168.200.14
14
```

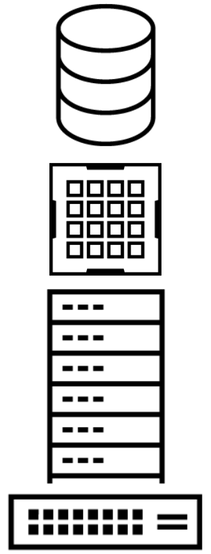
Loop

```
1   ---
2   - hosts: leaf
3     name: TOR
4     become: yes
5     gather_facts: no
6     tasks:
7       - name: copy eni
8         copy:
9           src: /home/cumulus/ON-15/step-06/{{ inventory_hostname }}-if
10          dest: /etc/network/interfaces
11      - name: activate changes on leaf01
12        shell: /sbin/ifreload -a
13
14      - name: copy frr
15        copy:
16          src: /home/cumulus/ON-15/step-06/{{ inventory_hostname }}-frr
17          dest: /etc/frr/frr.conf
18      - name: reload frr
19        ansible.builtin.shell: systemctl reload frr
20
```



standard modules

copy



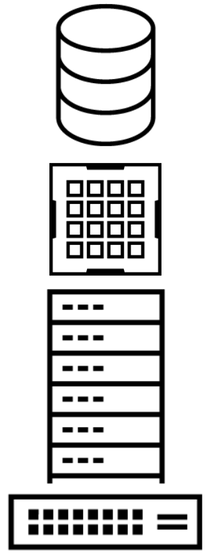
ORCHESTRATION

Step-06c: EVPN-MH (Layer3)

```
1   ---
2   - hosts: leaf01
3     name: task-step
4     become: yes
5     gather_facts: no
6
7     tasks:
8       - name: render file via template
9         template:
10          src: /home/cumulus/ON-15/step-06c/templates/leaf01-if.j2
11          dest: /etc/network/interfaces
12      - name: kick it
13        ansible.builtin.shell: /sbin/ifreload -a
14
15      - name: copy frr
16        copy:
17          src: /home/cumulus/ON-15/step-06c/leaf01-frr
18          dest: /etc/frr/frr.conf
19      - name: reload frr
20        ansible.builtin.shell: systemctl reload frr
21
22  # ************************************************************
23  # *****             Don't change anything below            *****
24  # ************************************************************
25
26  - name: Include the play for all other switches than leaf01
27    import_playbook: /home/cumulus/ON-15/step-06c/step-06.yaml
28
```

standard modules

copy



SUMMARY

- We need both an Orchestration tool/skill-set and a networking product and skill set.
- Configuration of Networking products can be conducted via IP(ROUTE2) commands, or more likely for most via rendered files or a CLI (NVUE).