The bridge to possible

CiscoLive: Brought to You By Cisco NSO and Cisco Modeling Labs

Joe Clarke, Distinguished Engineer

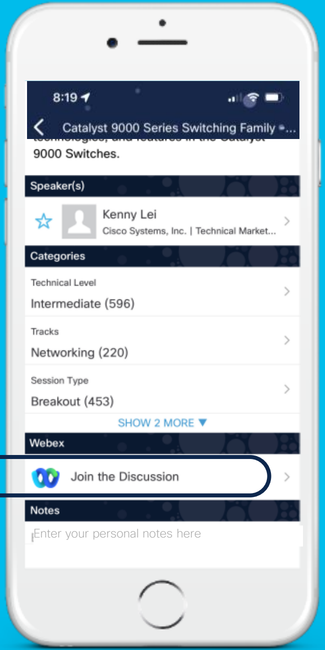
Cisco Webex App

Questions?

Use Cisco Webex App to chat
with the speaker after the session

How

1. Find this session in the Cisco Live Mobile App
2. Click “Join the Discussion”
3. Install the Webex App or go directly to the Webex space
4. Enter messages/questions in the Webex space



8:19

Catalyst 9000 Series Switching Family ...

9000 Switches.

Speaker(s)

Kenny Lei
Cisco Systems, Inc. | Technical Market...

Categories

Technical Level
Intermediate (596)

Tracks
Networking (220)

Session Type
Breakout (453)

SHOW 2 MORE ▼

Webex

Join the Discussion

Notes

Enter your personal notes here

Webex spaces will be moderated
until February 24, 2023.

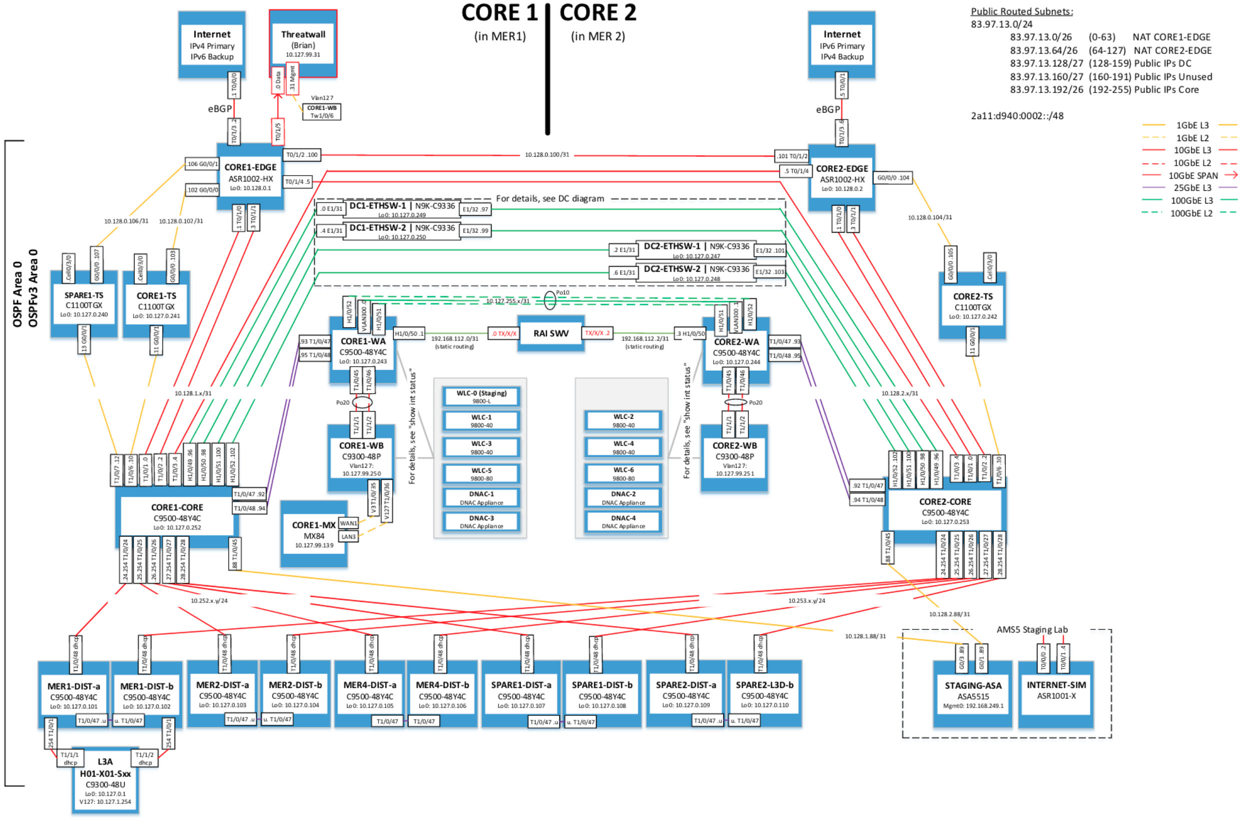
Abstract

In order to deliver a world-class event, we need a world-class network. The network that runs the event is staged ahead of time, but only in two weeks. It has to provide robust, stable services for thousands of attendees, and it only has a small window to make the best possible impression. Oh, and like all dynamic, agile environments, requests for changes come in beyond the last minute. So how do you build a network that you know will fit the bill? This session will deep-dive into the solution used to design, build, and test the data centre network here at CiscoLive. This DC infrastructure is driven by Cisco NSO service definitions and modeled ahead of time using Cisco Modeling Labs. Attendees will see how it was built and why.

What We’ve Been Looking For

We’re Getting There…

The Cisco Live Europe Network



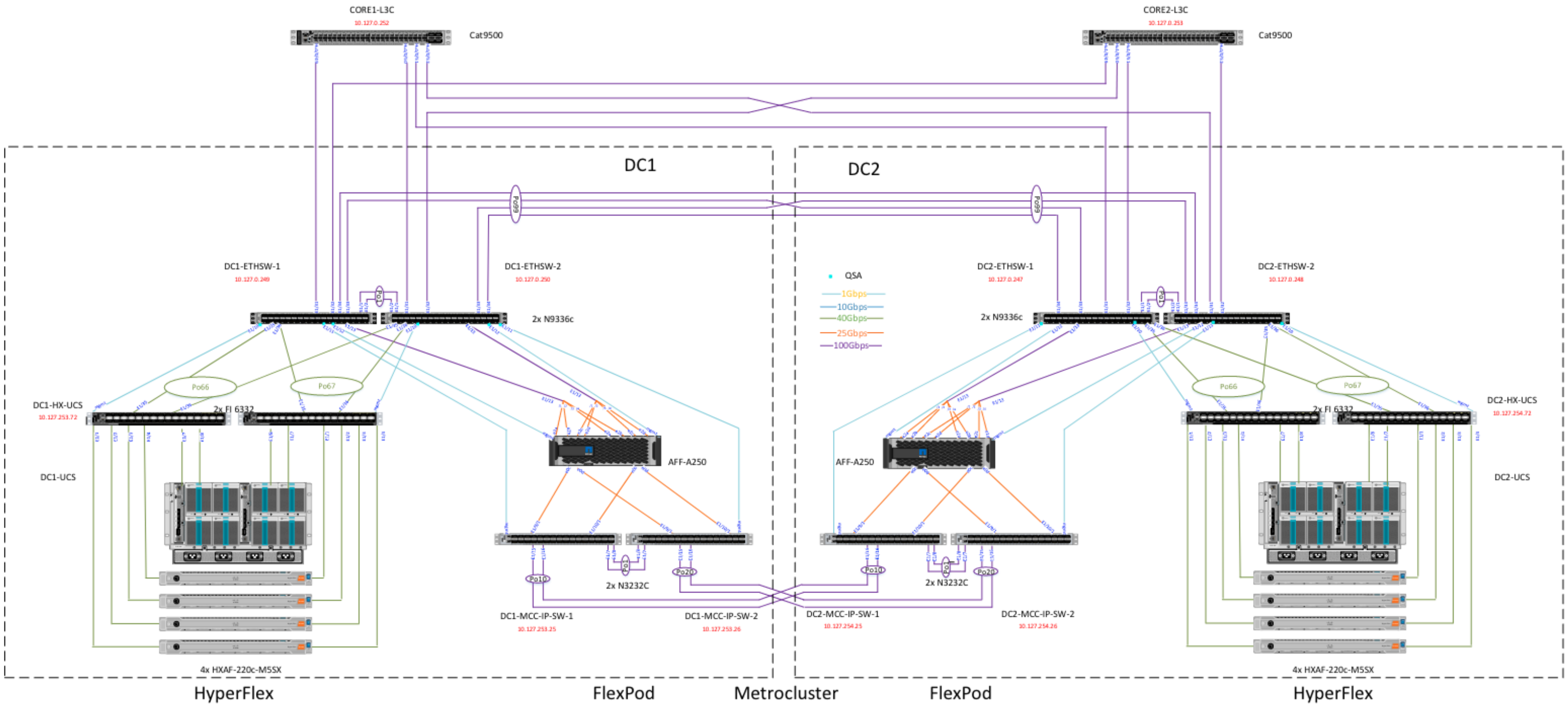
Public Routed Subnets:

83.97.13.0/24

83.97.13.0/26	(0-63)	NAT CORE1-EDGE
83.97.13.64/26	(64-127)	NAT CORE2-EDGE
83.97.13.128/27	(128-159)	Public IPs DC
83.97.13.160/27	(160-191)	Public IPs Unused
83.97.13.192/26	(192-255)	Public IPs Core

2a11:d940:0002::/48

The Data Centre Network



So Why Are You Here?

While this session is focused on what was done for the CiscoLive data centre, I'll wager…

[Insert Critical Service Here]

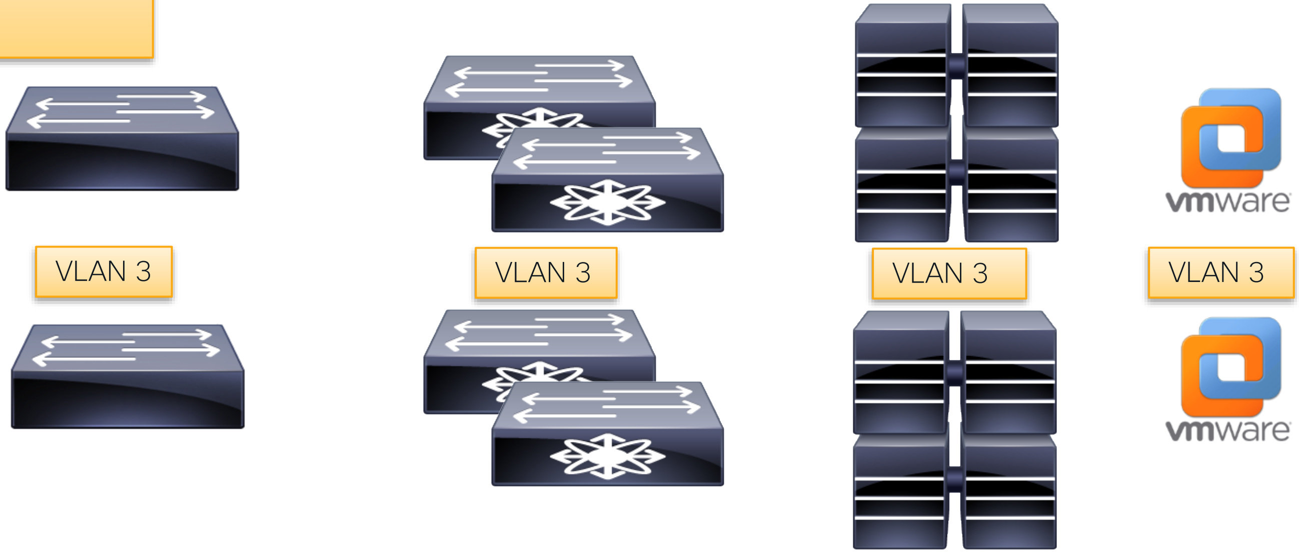
Agenda

- Introduction
  - Our Problem
  - Cisco NSO
  - Cisco Modeling Labs
- Building the Data Centre Model
- Testing Things Out
- Looking Forward to 2024

Our Problem

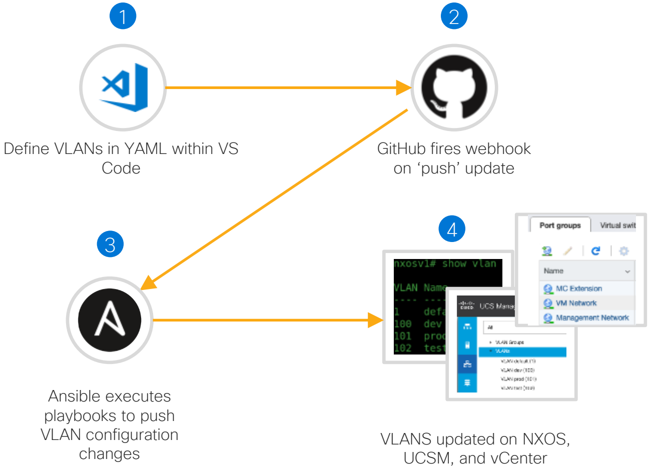
Event (All?) Networks Are Dynamic

No plan survives contact with production…



Solve It With Automation

Approach From 2020



1 Define VLANs in YAML within VS Code

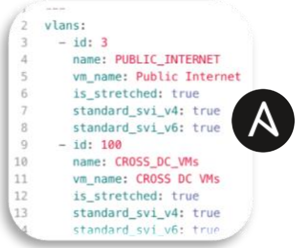
2 GitHub fires webhook on ‘push’ update

3 Ansible executes playbooks to push VLAN configuration changes

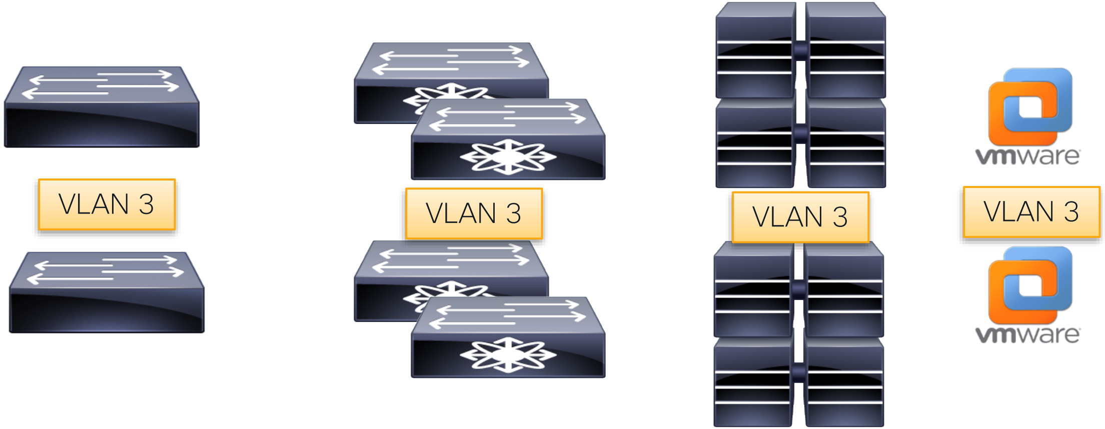
4 VLANS updated on NXOS, UCSM, and vCenter

```
nxosvl# show vlan

VLAN Name
---- ----
1    defa
100  dev
101  proc
102  test
```



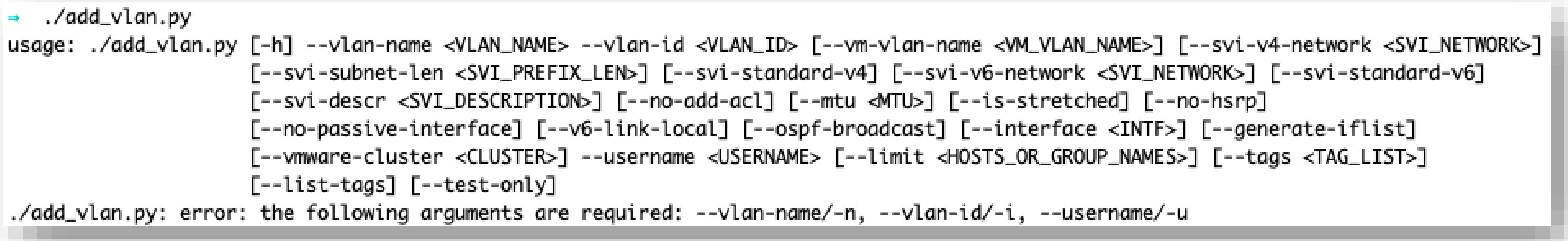
```yaml
---
vlans:
  - id: 3
    name: PUBLIC_INTERNET
    vm_name: Public Internet
    is_stretched: true
    standard_svi_v4: true
    standard_svi_v6: true
  - id: 100
    name: CROSS_DC_VMs
    vm_name: CROSS DC VMs
    is_stretched: true
    standard_svi_v4: true
    standard_svi_v6: true
```



Code available from https://marcuscom.com/git/MarcusCom/ciscolive/src/master/automation/cleu-ansible-n9k

Drawbacks With This First Approach

- While it worked well, it wasn’t so intuitive for network engineers, even through the CLI



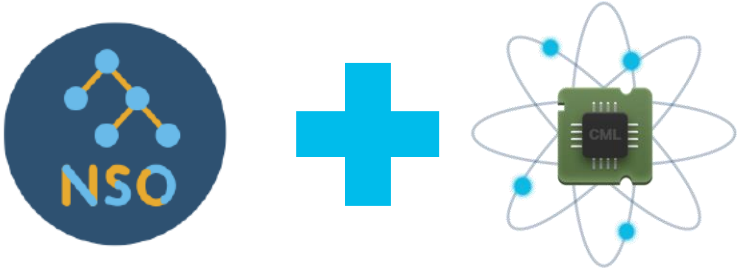
```
➜  ./add_vlan.py
usage: ./add_vlan.py [-h] --vlan-name <VLAN_NAME> --vlan-id <VLAN_ID> [--vm-vlan-name <VM_VLAN_NAME>] [--svi-v4-network <SVI_NETWORK>]
                     [--svi-subnet-len <SVI_PREFIX_LEN>] [--svi-standard-v4] [--svi-v6-network <SVI_NETWORK>] [--svi-standard-v6]
                     [--svi-descr <SVI_DESCRIPTION>] [--no-add-acl] [--mtu <MTU>] [--is-stretched] [--no-hsrp]
                     [--no-passive-interface] [--v6-link-local] [--ospf-broadcast] [--interface <INTF>] [--generate-iflist]
                     [--vmware-cluster <CLUSTER>] --username <USERNAME> [--limit <HOSTS_OR_GROUP_NAMES>] [--tags <TAG_LIST>]
                     [--list-tags] [--test-only]
./add_vlan.py: error: the following arguments are required: --vlan-name/-n, --vlan-id/-i, --username/-u
```

- Required additional work for each dynamic feature that was needed (i.e., we needed playbooks and scripts for static routes, switchports, credentials, etc.)

- Lacked state for easy iteration, backouts, and re-deployments

Something New For 2021…er, 2022…er, 2023, yeah!

- The time away provided amble opportunity to think through something different
- Wanted to iterate and test as I went
- Began work in late 2021, settling on NSO for service development and a common interface to the data centre with CML as my initial testbed
- Is it perfect? No, but don’t let perfect be the enemy of good.



Cisco Network Services Orchestrator (NSO)

NSO will help a lot now, and even more later

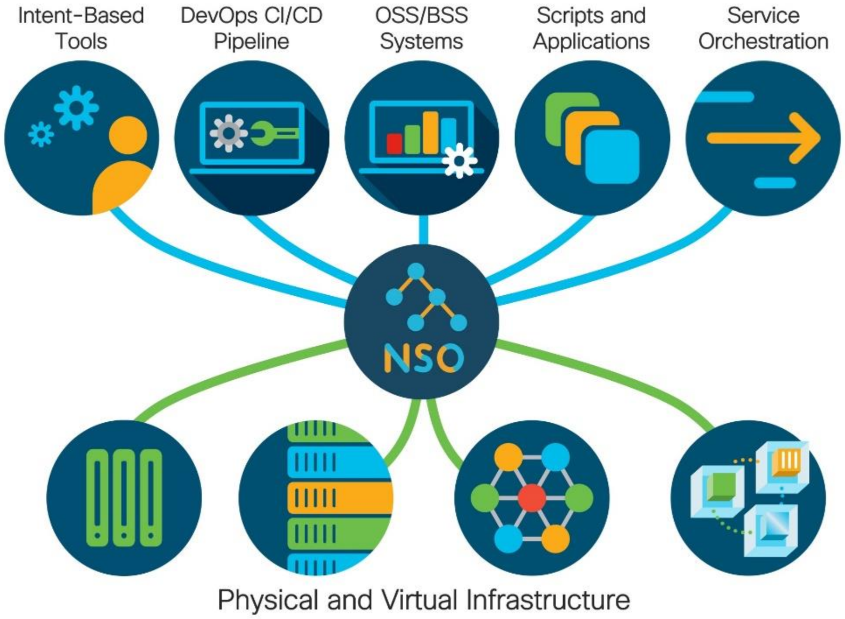
Today:
Network CLI

- Immediate benefits with familiar approach
- 2 fully functional CLI options
- Strict separation of operational and config data
- Perform operations on groups of devices
- Full AAA integration
- Database-style two-phase commit on changes

Tomorrow:
Network API

- Adopt over time as comfort and skills grow
- Python is a good place to start
- Multiple language bindings available
- Eventually integrate your own toolchain

Cisco NSO is a bridge

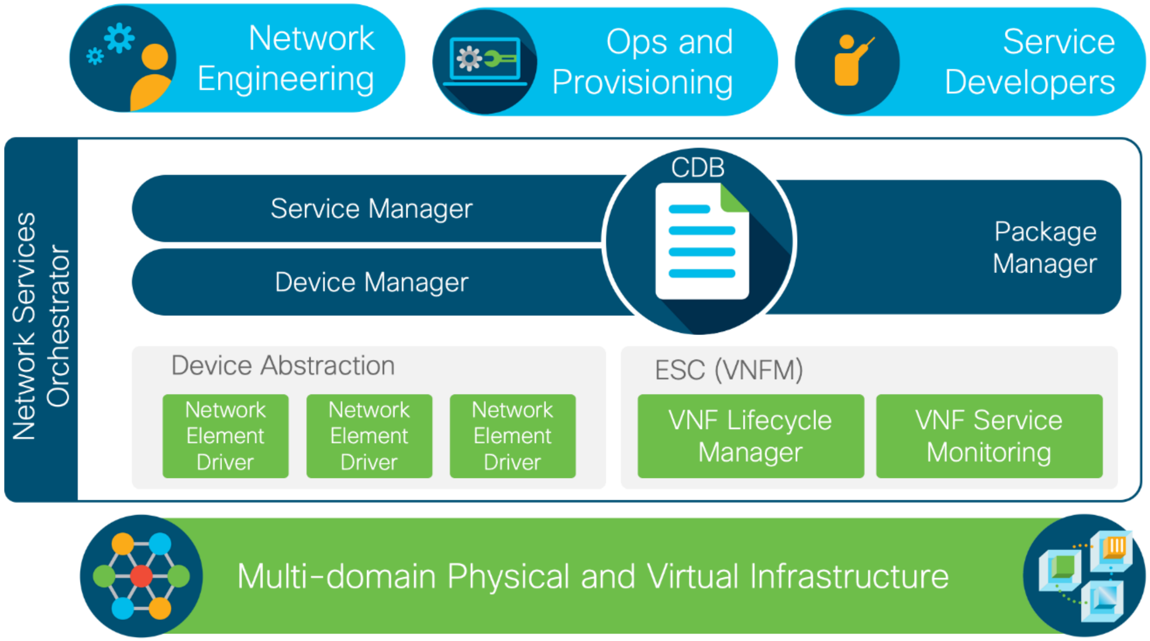


Between people that build services and ones that operate infrastructure

Across different domains and vendors

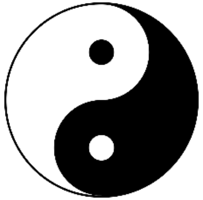
Over both physical and virtual infrastructure

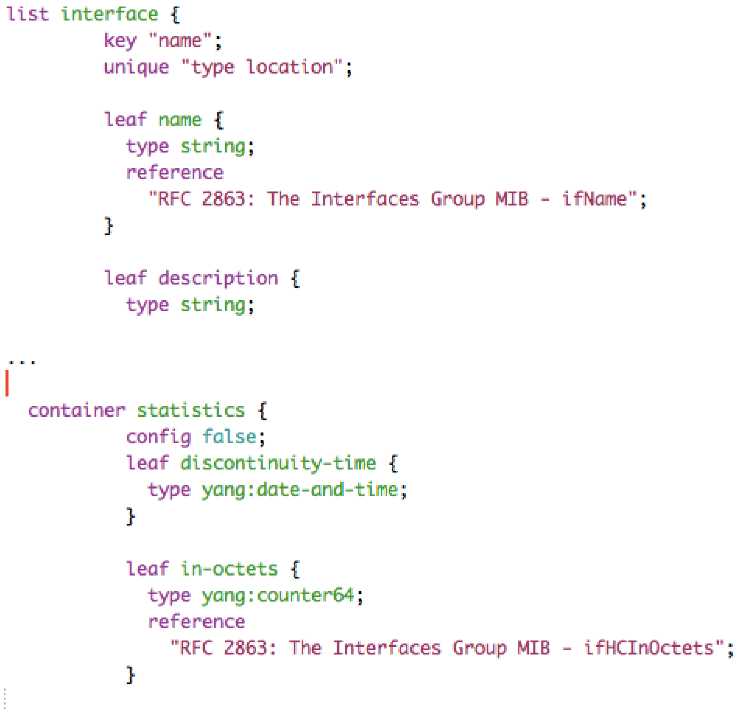
NSO Architecture



- Multivendor abstraction through Network Element Drivers (NEDs)
- Single datastore for all network elements under management
- Multiple interfaces including CLI, REST, Java Python
- Templates and compliance reporting
- YANG-based configuration schema

A Few Words About YANG





```
list interface {
         key "name";
         unique "type location";

         leaf name {
           type string;
           reference
             "RFC 2863: The Interfaces Group MIB - ifName";
         }

         leaf description {
           type string;

...

  container statistics {
           config false;
           leaf discontinuity-time {
             type yang:date-and-time;
           }

           leaf in-octets {
             type yang:counter64;
             reference
               "RFC 2863: The Interfaces Group MIB - ifHCInOctets";
           }
```

Data modeling language built for network configuration

Human readable

Hierarchical configuration

Extensible through augmentation

Reusable types and groupings

Modular and expressive

Standards-based (defined in RFC7950)

Getting NSO Files

- Available FREE on DevNet for non-production use
  - https://developer.cisco.com/docs/nso/#!getting-nso
- What you'll get
  - NSO installation file for Intel Mac or Linux
  - Latest NEDs for Cisco platforms (IOS, XR, NX-OS, ASA)

Cisco Modeling Labs

Cisco Modeling Labs

Your lab anywhere

Browser-based, point-and-click network design, config, testing, troubleshooting

Fully programmable network simulation and testing

Fast setup and tear-down

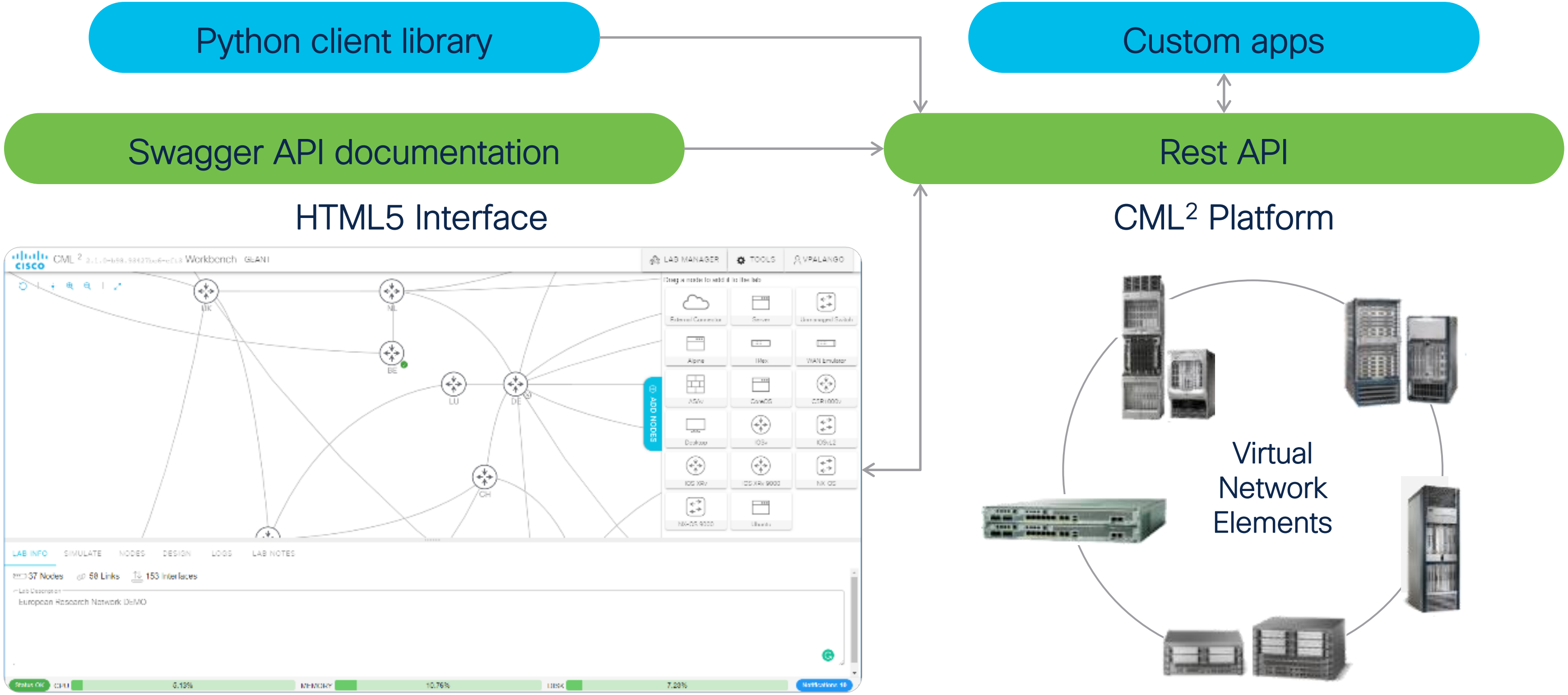
Labs are persistent between sessions

Seamless connectivity with physical networks and other simulations

API-first approach

Ships with Cisco images and open-source images

Architecture Overview



Python client library

Custom apps

Swagger API documentation

Rest API

HTML5 Interface

CML^2 Platform

Virtual Network Elements

Key Features

Labs = Isolated virtual networks

Configurable network connectivity

Console multiplexing

Flexible console access

Live topology modification – drag, drop, and wire networks in a running simulation

Device persistence – just like shutting down a real router

Admin control of all labs

Ability to define and import third-party device types that support the Linux KVM hypervisor

Server-side labs with an easy-to-use HTML5 front end

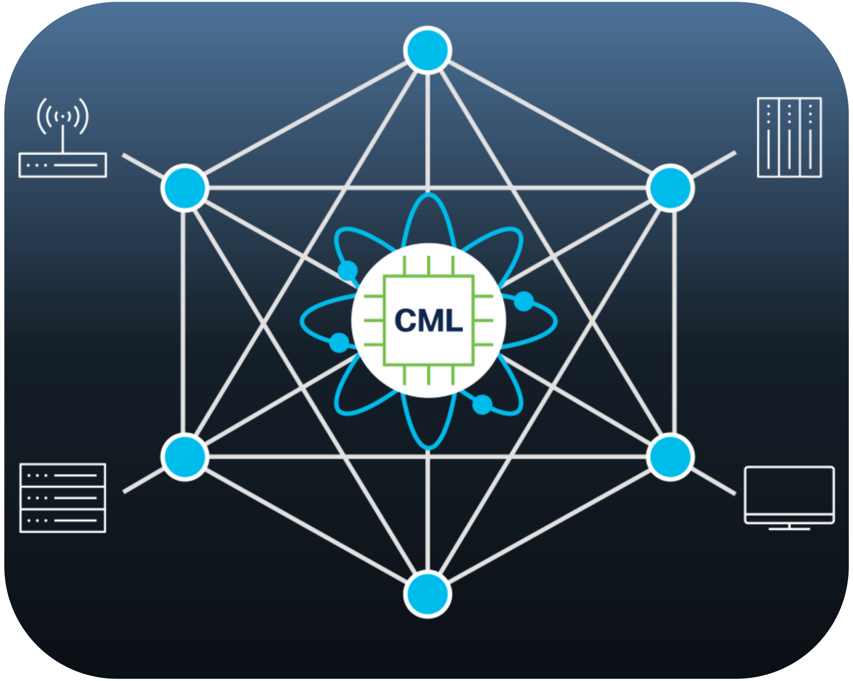
{ REST:API }

API First Design

Swagger interface

Python client library

Getting the CML Bits



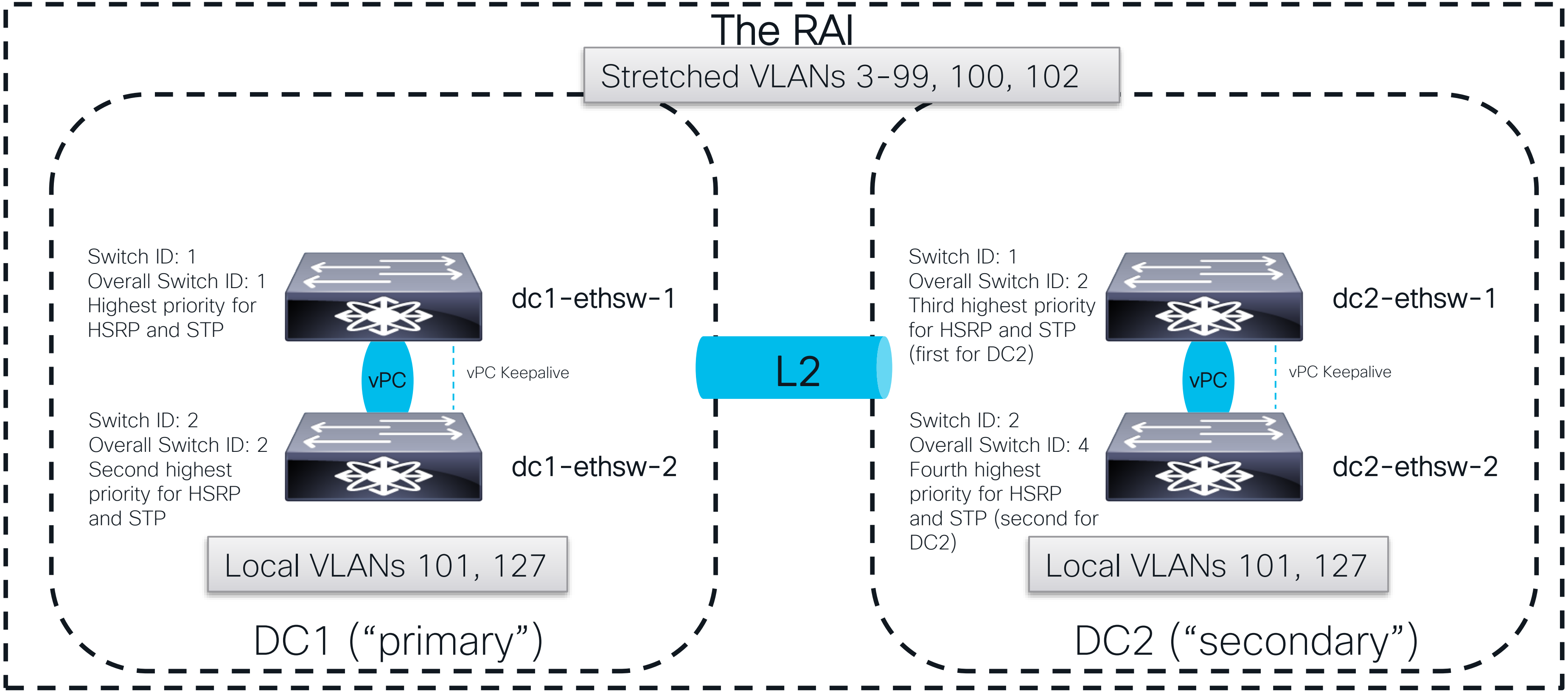
- Obtaining: https://developer.cisco.com/docs/modeling-labs/#!downloading-files-for-cml-installation

- Docs: https://developer.cisco.com/docs/modeling-labs/#!introduction

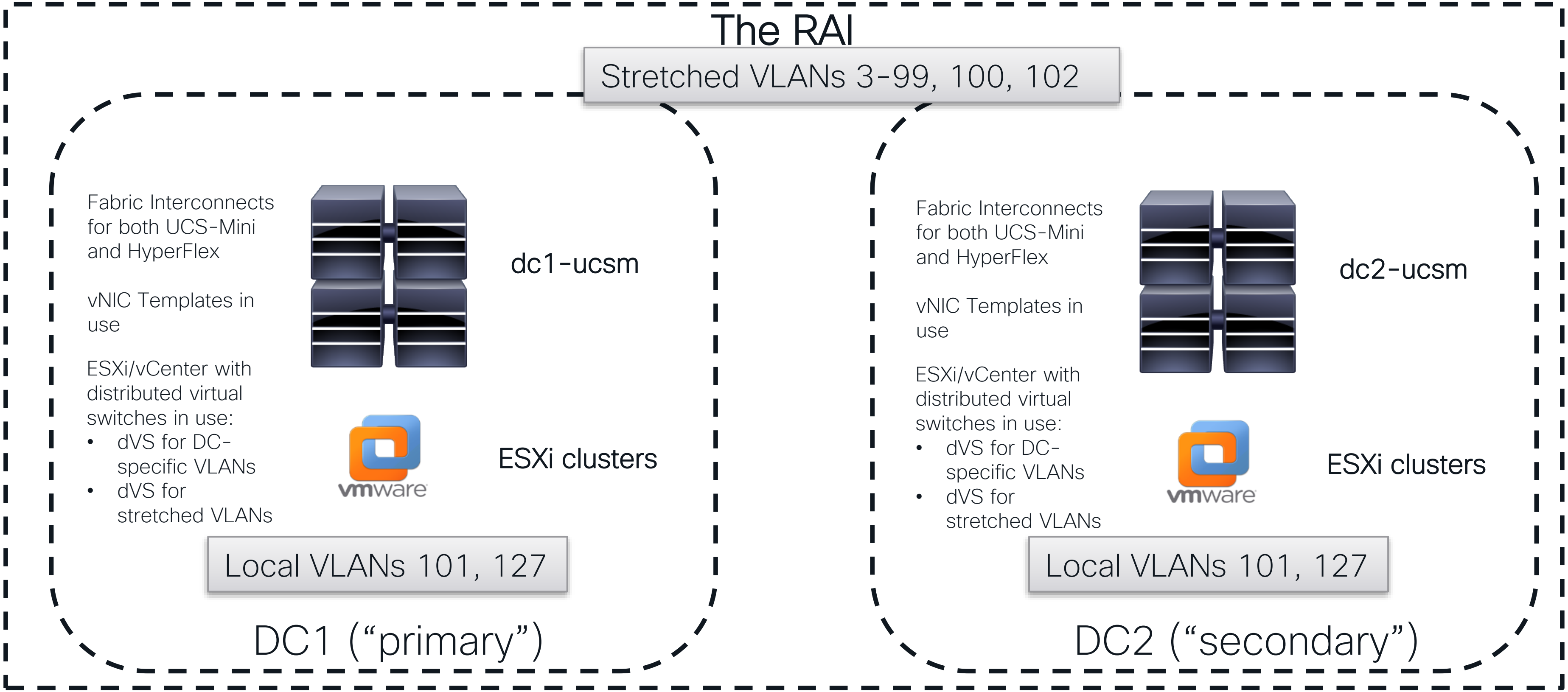
- Sandbox: https://devnetsandbox.cisco.com/RM/Topology (Reserve “Cisco Modeling Labs Enterprise”)

Building the Data Centre Model

Some More About Our DC Design (Network)



Some More About Our DC Design (Compute)



The RAI

Stretched VLANs 3-99, 100, 102

Fabric Interconnects for both UCS-Mini and HyperFlex

vNIC Templates in use

ESXi/vCenter with distributed virtual switches in use:

- dVS for DC-specific VLANs
- dVS for stretched VLANs

dc1-ucsm

ESXi clusters

Local VLANs 101, 127

DC1 (“primary”)

Fabric Interconnects for both UCS-Mini and HyperFlex

vNIC Templates in use

ESXi/vCenter with distributed virtual switches in use:

- dVS for DC-specific VLANs
- dVS for stretched VLANs

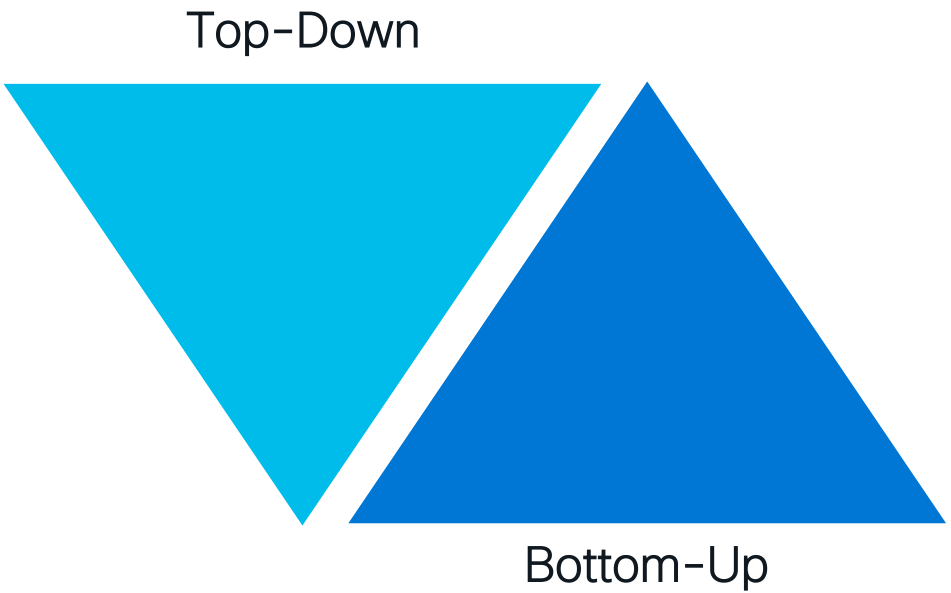
dc2-ucsm

ESXi clusters

Local VLANs 101, 127

DC2 (“secondary”)

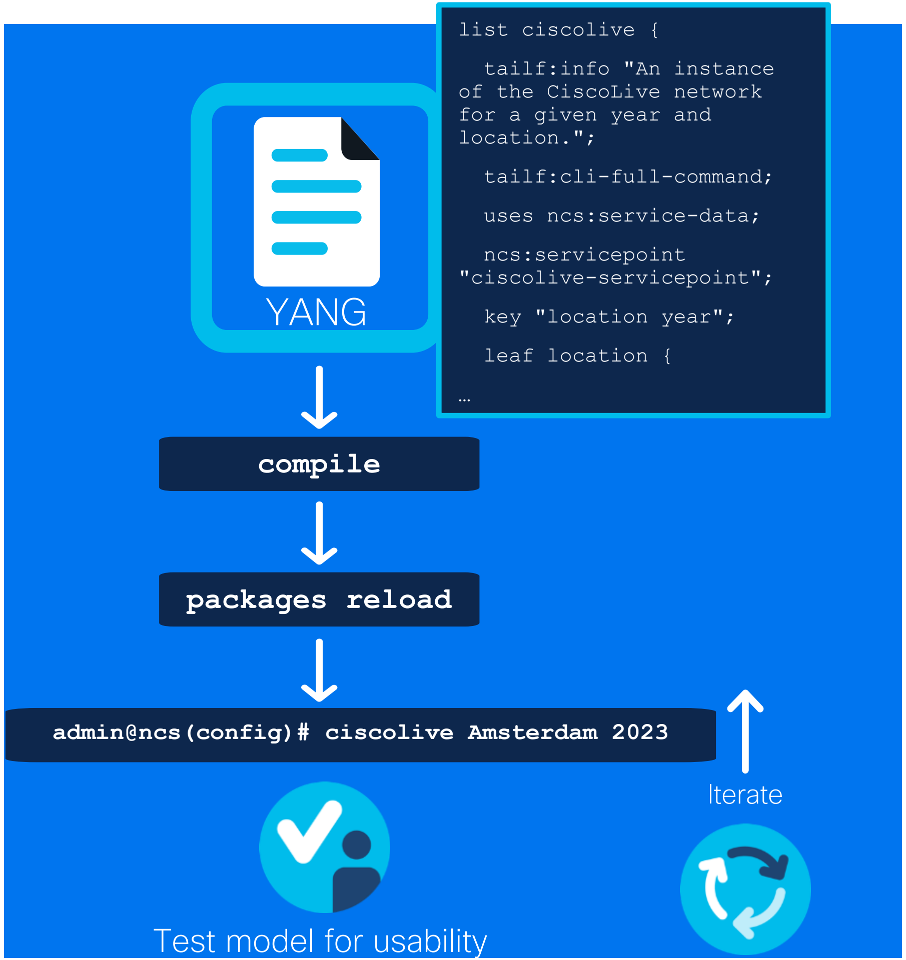
Choosing the Service Design Approach



Ultimately, this is choosing the user experience, too.

Top-Down Approach

- Good for green-field services
- One thinks of the overall model and flow first
- From that the templates (i.e., southbound* is developed)



```
list ciscolive {

  tailf:info "An instance
of the CiscoLive network
for a given year and
location.";

  tailf:cli-full-command;

  uses ncs:service-data;

  ncs:servicepoint
"ciscolive-servicepoint";

  key "location year";

  leaf location {

…
```

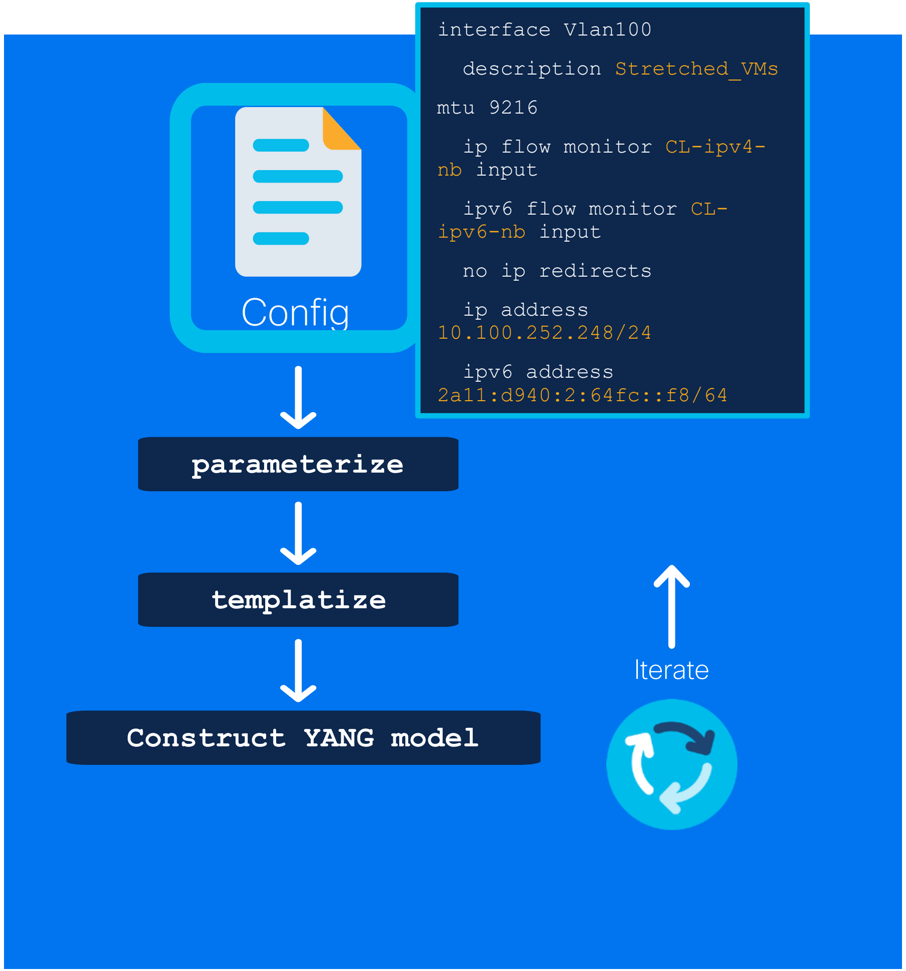
```
admin@ncs(config)# ciscolive Amsterdam 2023
```

Test model for usability

* **Southbound:** meaning towards the end devices

Bottom-Up Approach

- Work from the templates or the CLI and derive the model from that
- Effective when you have a configuration or service that you want to automate



```
interface Vlan100
  description Stretched_VMs
mtu 9216
  ip flow monitor CL-ipv4-nb input
  ipv6 flow monitor CL-ipv6-nb input
  no ip redirects
  ip address 10.100.252.248/24
  ipv6 address 2a11:d940:2:64fc::f8/64
```

CiscoLive Met in the Middle

- Had an idea of what I wanted to be for the experience
- Knew which parameters I wanted to be variable and which I wanted to be static
- Allowed for some new service elements (e.g., HSRP security)

- Used existing configs from previous years to inform the model
- Provided checks and balances to ensure nothing was left out
- Able to identify technical debt

NSO Components Used

- Cisco NSO 5.6.0 ← Started here
- Ultimately landed on **NSO 6.0.0** for CLEU 2023
- NEDs
  - Cisco-NX 5.23.7 : For N9K switches
  - Cisco-UCS 3.4.4 : For UCS Manager
  - VMware-vSphere 3.3.5 : For vCenter
- **NOTE:** Initially lacked some IPv6 features (OSPFv3 and neighbor discovery config features); thanks to NSO TAC/dev for quick turn-around 🎉

Getting Started

- The service *package* for CiscoLive uses multiple templates (think interface config, VLAN config, router config, etc.) so Python is required

  ```
  ncs-make-package --service-skeleton python-and-template
  ciscolive
  ```

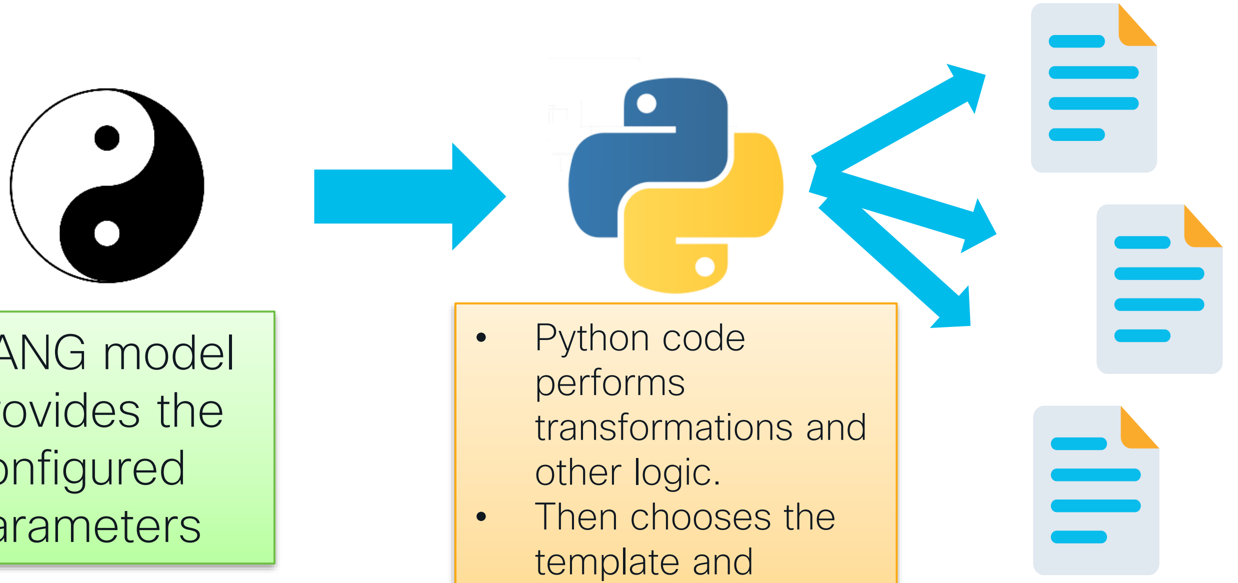
- Iteration is key so did all work in git (plus you can have the code 😀)
- For my `ncs.conf`, I added one bit (for Cisco-style CLI):

```
<cli>

  <style>c</style>

</cli>
```

How The Python Package Works



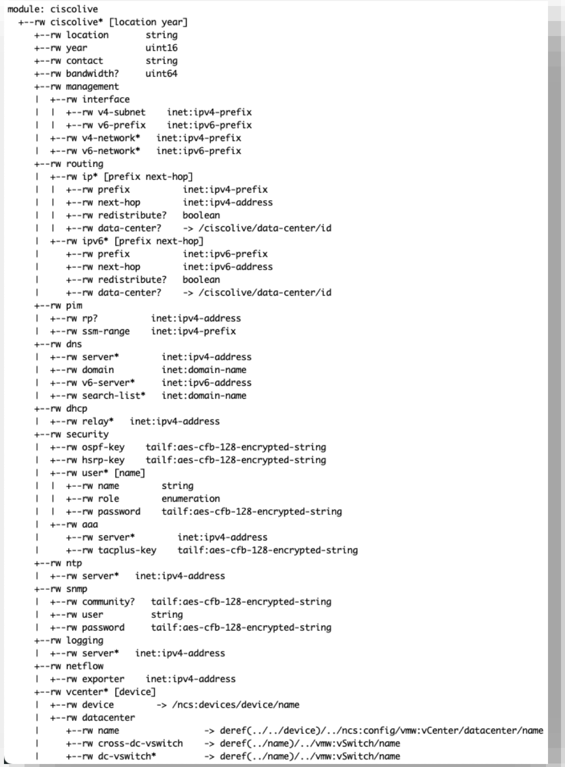
YANG model provides the configured parameters

- Python code performs transformations and other logic.
- Then chooses the template and applies it with the correct parameters

XML templates are transformed to southbound CLI

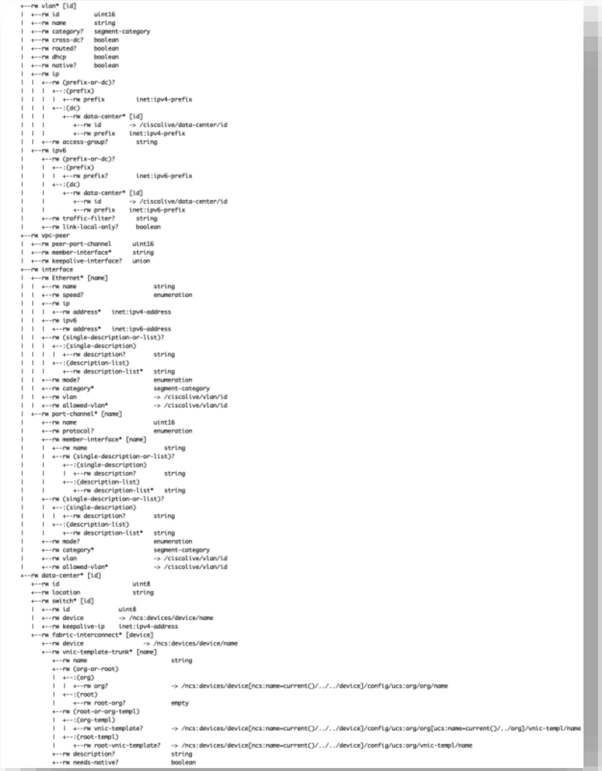
```
interface Vlan 127
 description DC1-MGMT
 ip address
10.127.253.248/24
 ip router ospf 1 area
0.0.0.0
…
```

The Model Tree



```
module: ciscolive
  +--rw ciscolive* [location year]
     +--rw location         string
     +--rw year             uint16
     +--rw contact          string
     +--rw bandwidth?       uint64
     +--rw management
     |  +--rw interface
     |  |  +--rw v4-subnet    inet:ipv4-prefix
     |  |  +--rw v6-prefix    inet:ipv6-prefix
     |  +--rw v4-network*   inet:ipv4-prefix
     |  +--rw v6-network*   inet:ipv6-prefix
     +--rw routing
     |  +--rw ip* [prefix next-hop]
     |  |  +--rw prefix           inet:ipv4-prefix
     |  |  +--rw next-hop         inet:ipv4-address
     |  |  +--rw redistribute?    boolean
     |  |  +--rw data-center?     -> /ciscolive/data-center/id
     |  +--rw ipv6* [prefix next-hop]
     |     +--rw prefix           inet:ipv6-prefix
     |     +--rw next-hop         inet:ipv6-address
     |     +--rw redistribute?    boolean
     |     +--rw data-center?     -> /ciscolive/data-center/id
     +--rw pim
     |  +--rw rp?          inet:ipv4-address
     |  +--rw ssm-range    inet:ipv4-prefix
     +--rw dns
     |  +--rw server*        inet:ipv4-address
     |  +--rw domain         inet:domain-name
     |  +--rw v6-server*     inet:ipv6-address
     |  +--rw search-list*   inet:domain-name
     +--rw dhcp
     |  +--rw relay*   inet:ipv4-address
     +--rw security
     |  +--rw ospf-key    tailf:aes-cfb-128-encrypted-string
     |  +--rw hsrp-key    tailf:aes-cfb-128-encrypted-string
     |  +--rw user* [name]
     |  |  +--rw name        string
     |  |  +--rw role        enumeration
     |  |  +--rw password    tailf:aes-cfb-128-encrypted-string
     |  +--rw aaa
     |     +--rw server*        inet:ipv4-address
     |     +--rw tacplus-key    tailf:aes-cfb-128-encrypted-string
     +--rw ntp
     |  +--rw server*   inet:ipv4-address
     +--rw snmp
     |  +--rw community?   tailf:aes-cfb-128-encrypted-string
     |  +--rw user         string
     |  +--rw password     tailf:aes-cfb-128-encrypted-string
     +--rw logging
     |  +--rw server*   inet:ipv4-address
     +--rw netflow
     |  +--rw exporter    inet:ipv4-address
     +--rw vcenter* [device]
     |  +--rw device       -> /ncs:devices/device/name
     |  +--rw datacenter
     |     +--rw name             -> deref(../../device)/../ncs:config/vmw:vCenter/datacenter/name
     |     +--rw cross-dc-vswitch -> deref(../name)/../vmw:vSwitch/name
     |     +--rw dc-vswitch*      -> deref(../name)/../vmw:vSwitch/name
```

- Generated with `pyang -f tree`
- We'll discuss some of these elements



```
+--rw vlan* [id]
|  +--rw id          uint16
|  +--rw name        string
|  +--rw category?   segment-category
|  +--rw cross-dc?   boolean
|  +--rw routed?     boolean
|  +--rw dhcp        boolean
|  +--rw native?     boolean
|  +--rw ip
|  |  +--rw (prefix-or-dc)?
|  |  |  +--:(prefix)
|  |  |  |  +--rw prefix         inet:ipv4-prefix
|  |  |  +--:(dc)
|  |  |     +--rw data-center* [id]
|  |  |        +--rw id        -> /ciscolive/data-center/id
|  |  |        +--rw prefix    inet:ipv4-prefix
|  |  +--rw access-group?        string
|  +--rw ipv6
|     +--rw (prefix-or-dc)?
|     |  +--:(prefix)
|     |  |  +--rw prefix?        inet:ipv6-prefix
|     |  +--:(dc)
|     |     +--rw data-center* [id]
|     |        +--rw id        -> /ciscolive/data-center/id
|     |        +--rw prefix    inet:ipv6-prefix
|     +--rw traffic-filter?      string
|     +--rw link-local-only?     boolean
+--rw vpc-peer
|  +--rw peer-port-channel      uint16
|  +--rw member-interface*      string
|  +--rw keepalive-interface?   union
+--rw interface
|  +--rw Ethernet* [name]
|  |  +--rw name                          string
|  |  +--rw speed?                        enumeration
|  |  +--rw ip
|  |  |  +--rw address*   inet:ipv4-address
|  |  +--rw ipv6
|  |  |  +--rw address*   inet:ipv6-address
|  |  +--rw (single-description-or-list)?
|  |  |  +--:(single-description)
|  |  |  |  +--rw description?        string
|  |  |  +--:(description-list)
|  |  |     +--rw description-list*   string
|  |  +--rw mode?                         enumeration
|  |  +--rw category*                     segment-category
|  |  +--rw vlan                          -> /ciscolive/vlan/id
|  |  +--rw allowed-vlan*                 -> /ciscolive/vlan/id
|  +--rw port-channel* [name]
|     +--rw name                          uint16
|     +--rw protocol?                     enumeration
|     +--rw member-interface* [name]
|     |  +--rw name                          string
|     |  +--rw (single-description-or-list)?
|     |     +--:(single-description)
|     |     |  +--rw description?        string
|     |     +--:(description-list)
|     |        +--rw description-list*   string
|     +--rw (single-description-or-list)?
|     |  +--:(single-description)
|     |  |  +--rw description?        string
|     |  +--:(description-list)
|     |     +--rw description-list*   string
|     +--rw mode?                         enumeration
|     +--rw category*                     segment-category
|     +--rw vlan                          -> /ciscolive/vlan/id
|     +--rw allowed-vlan*                 -> /ciscolive/vlan/id
+--rw data-center* [id]
   +--rw id                       uint8
   +--rw location                 string
   +--rw switch* [id]
   |  +--rw id             uint8
   |  +--rw device         -> /ncs:devices/device/name
   |  +--rw keepalive-ip   inet:ipv4-address
   +--rw fabric-interconnect* [device]
      +--rw device                 -> /ncs:devices/device/name
      +--rw vnic-template-trunk* [name]
         +--rw name                          string
         +--rw (org-or-root)
         |  +--:(org)
         |  |  +--rw org?                    -> /ncs:devices/device[ncs:name=current()/../../device]/config/ucs:org/org/name
         |  +--:(root)
         |     +--rw root-org?               empty
         +--rw (root-or-org-templ)
         |  +--:(org-templ)
         |  |  +--rw vnic-template?          -> /ncs:devices/device[ncs:name=current()/../../device]/config/ucs:org/org[ucs:name=current()/../org]/vnic-templ/name
         |  +--:(root-templ)
         |     +--rw root-vnic-template?     -> /ncs:devices/device[ncs:name=current()/../../device]/config/ucs:org/vnic-templ/name
         +--rw description?                  string
         +--rw needs-native?                 boolean
```

Spinning Up a New ”ciscolive”

Tree

```
module: ciscolive
  +--rw ciscolive* [location year]
     +--rw location       string
     +--rw year           uint16
```

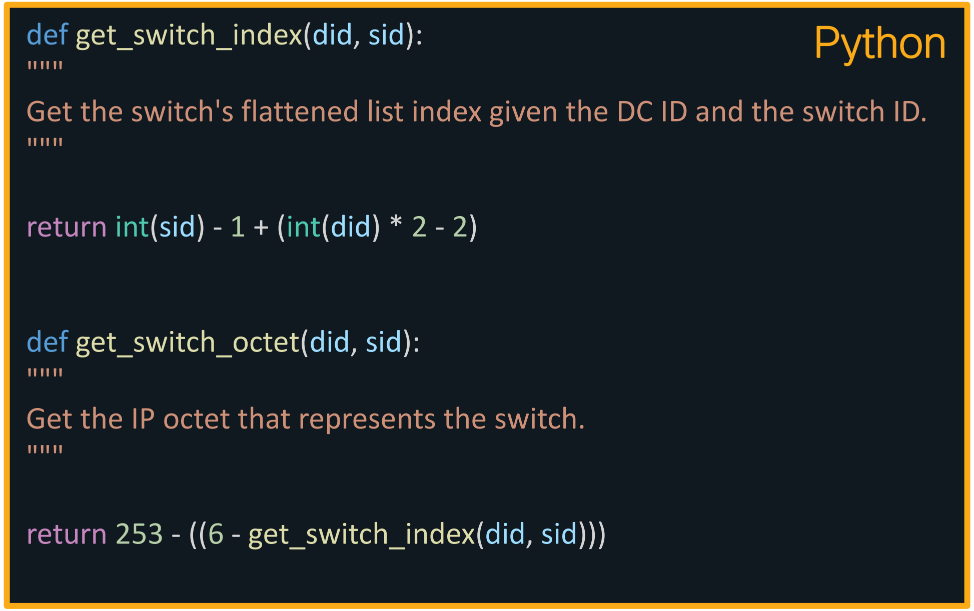
```
list ciscolive {
    tailf:info "An instance of the CiscoLive network for a given
year and location.";
    tailf:cli-full-command;
    uses ncs:service-data;
    ncs:servicepoint "ciscolive-servicepoint";
    key "location year";
    leaf location {
        tailf:info "Location of this instance of the
CiscoLive network.";
        type string;
    }
    leaf year {
        tailf:info "Year for this instance of the CiscoLive DC
network.";
        type uint16;
     }
```

YANG

- NSO expects service *instances* to be a list
- Each “ciscolive” is indexed using its location and year
- The `tailf:cli-full-command` extension is a Cisco nicety which lets one enter “ciscolive” sub-mode with a command like `ciscolive Amsterdam 2023`
- The `tailf:info` extension provides CLI context-sensitive help

Choosing IDs and Octets

- Some small Python snippets are used to determine its position in a list object (also used for calculating HSRP and STP priorities)
- The core architects like(d) .254/::fe for the [HSRP] gateways, so also grab high values for per-switch octets
- sid == switch ID (1 or 2)
- did == DC ID (1 or 2)
  - dc1-ethsw-1: [1 – 1 + (1 * 2 – 2) = **0**]
  - [253 – ((6 – 0)) = **247**]



Python

```python
def get_switch_index(did, sid):
"""
Get the switch's flattened list index given the DC ID and the switch ID.
"""

return int(sid) - 1 + (int(did) * 2 - 2)


def get_switch_octet(did, sid):
"""
Get the IP octet that represents the switch.
"""

return 253 - ((6 - get_switch_index(did, sid)))
```

HSRP and STP

- dc1-ethsw-1 should have lowest STP priority and highest HSRP priority
- Each subsequent switch should follow in suit
- STP:
  - dc1-ethsw-1: 0 * 4096 = **0**
  - dc2-ethsw-1: 2 * 4096 = **8192**
- HSRP:
  - dc1-ethsw-2: 105 – 1 = **104**
  - dc2-ethsw-2: 105 – 3 = **102**

```
# Set a spanning-tree priority based on switch and DC IDs.
stp_prio = get_switch_index(dc.id, switch.id) * 4096
…

# Determine HSRP priority based on switch ID and DC ID
svi_vars.add("HSRP_PRIORITY", 105 - get_switch_index(dc.id,
switch.id))
```

Python

With Respect to Passwords…

```
show running-config
ciscolive Amsterdam 2023
snmp
```

YANG

```
container snmp {
  leaf community {
    tailf:info "SNMP community string (read-only)";
    type tailf:aes-cfb-128-encrypted-string;
  }
  leaf password {
    tailf:info "SNMPv3 password.";
    type tailf:aes-cfb-128-encrypted-string;
    mandatory true;
  }
}
```

```
ciscolive Amsterdam 2023
 snmp
  community not-public
  user      CLEUR
  password really-secret
 !
!
```

```
ciscolive Amsterdam 2023
 snmp
  community $8$Ykq…
  user      CLEUR
  password $8$5IQ…
 !
```

Python

```
from _ncs import decrypt
…
trans = ncs.maagic.get_trans(root)
trans.maapi.install_crypto_keys()
…
snmp_vars.add("COMMUNITY",
decrypt(self.service.snmp.community))
```

Modeling VLANs

Tree

```
+--rw vlan* [id]
     |   +--rw id            uint16
     |   +--rw name          string
     |   +--rw category?     segment-
category
     |   +--rw cross-dc?     boolean
     |   +--rw routed?       boolean
     |   +--rw dhcp          boolean
     |   +--rw native?       boolean
     |   +--rw ip
…
     |   +--rw ipv6
…
```

- Each VLAN can be L2 only or routed (i.e., having an SVI)
- A VLAN can either be local to a DC or stretched (this mainly affects IP space)
- The `category` leaf is used to automatically group allowed VLANs on trunks

Segment Categories

- An enumeration typedef to better describe VLANs
- Tied to interfaces (interfaces can have multiple categories)

```
typedef segment-category {
  tailf:info "Usage category of a given
segment.";
  type enumeration {
    enum vm {
      tailf:info "Segment is used for virtual
machines.";
    }
    enum netapp {
      tailf:info "Segment is used for NetApp-
only traffic.";
    }
    enum fabric-interconnect {
      tailf:info "Segment is used for FI-only
traffic.";
    }
    enum peer {
      tailf:info "Segment is used for route
peering.";
    }
  }
}
```

YANG

Constructing the L2 VLAN Parameters for Switches and FIs

- Parameterize the VLAN ID
- When creating the "name" parameter, substitute "{dc}" and "{peer_dc}" strings with the DC ID and peer DC ID respectively
  - Convenient for those DC-local VLANs
  - Allows for a name like "MGMT-DC1" vs. simply "MGMT"

Python

```
for vlan in self.service.vlan:
    l2_vars.add("VLAN_ID", vlan.id)
    l2_vars.add("VLAN_NAME",
vlan.name.format(dc=dc.id,
peer_dc=get_peer_id(dc.id)))
```

Constructing the L2 VLAN Parameters for vCenter

- Like the network and compute sides
- PG or Port Group is more appropriate as a parameter name
- The correct dVS also needs to be specified
- Note: the category leaf is used here so that we only add VM-supporting VLANs

Python

```
if vlan.category != "vm":
   continue

vc_vars.add("VLAN_ID", vlan.id)
vc_vars.add("PG_NAME", vlan.name.format(dc=dc.id,
peer_dc=get_peer_id(dc.id)))

if vlan.cross_dc:
   vc_vars.add("VSWITCH",
vcenter.datacenter.cross_dc_vswitch)
else:
   vc_vars.add("VSWITCH",
list(vcenter.datacenter.dc_vswitch)[int(dc.id) - 1])
```

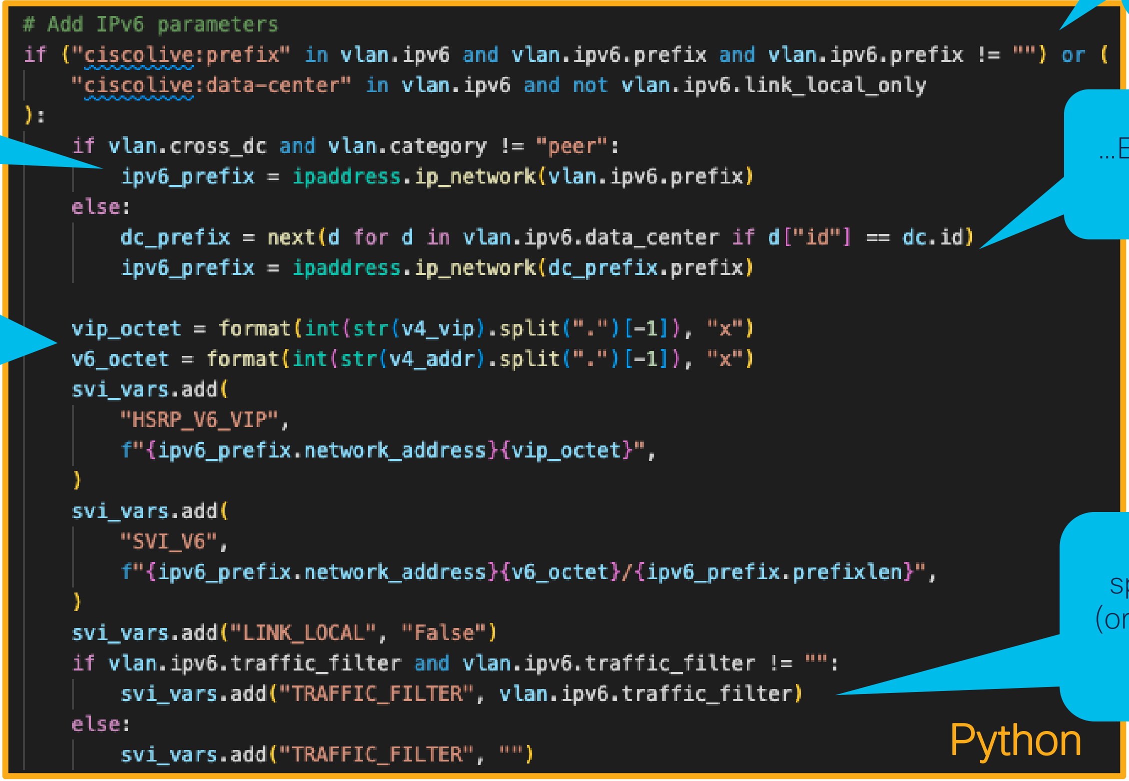
Constructing the L3 SVI Parameters

Note: for IPv6, an SVI doesn’t need a global address. We can use link-local only.

If the VLAN is a stretched VLAN get its single prefix

…Else get its per-DC prefix

Calculate the last octet for the SVI and for the HSRP VIP as the hex version of the v4 octet



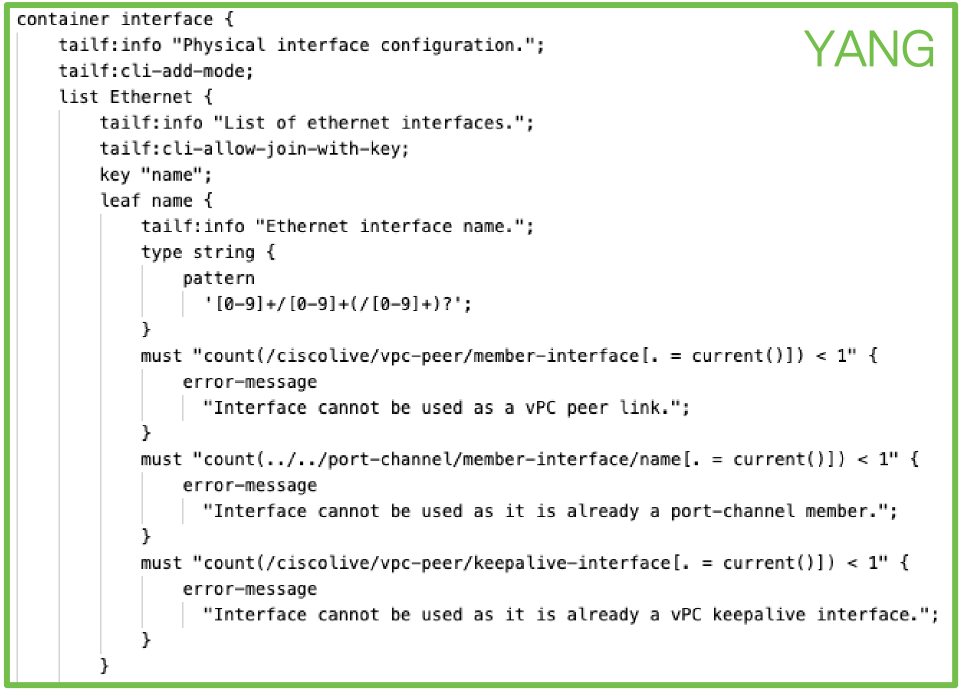
```
# Add IPv6 parameters
if ("ciscolive:prefix" in vlan.ipv6 and vlan.ipv6.prefix and vlan.ipv6.prefix != "") or (
    "ciscolive:data-center" in vlan.ipv6 and not vlan.ipv6.link_local_only
):
    if vlan.cross_dc and vlan.category != "peer":
        ipv6_prefix = ipaddress.ip_network(vlan.ipv6.prefix)
    else:
        dc_prefix = next(d for d in vlan.ipv6.data_center if d["id"] == dc.id)
        ipv6_prefix = ipaddress.ip_network(dc_prefix.prefix)

    vip_octet = format(int(str(v4_vip).split(".")[-1]), "x")
    v6_octet = format(int(str(v4_addr).split(".")[-1]), "x")
    svi_vars.add(
        "HSRP_V6_VIP",
        f"{ipv6_prefix.network_address}{vip_octet}",
    )
    svi_vars.add(
        "SVI_V6",
        f"{ipv6_prefix.network_address}{v6_octet}/{ipv6_prefix.prefixlen}",
    )
    svi_vars.add("LINK_LOCAL", "False")
    if vlan.ipv6.traffic_filter and vlan.ipv6.traffic_filter != "":
        svi_vars.add("TRAFFIC_FILTER", vlan.ipv6.traffic_filter)
    else:
        svi_vars.add("TRAFFIC_FILTER", "")
```

The model lets us specify a traffic-filter (or ACL), but those are not defined in the model

Modeling Ethernet Interfaces

- Each ethernet interface has the same config on all switches
- The must constraints ensure an interface isn't already used for vPC peer or part of a port-channel (those are handled differently in the model)
- The `tailf:cli-add-mode` extension allows a sub-mode for `interface Ethernet 1/1`
- The `tailf:cli-allow-join-with-key` extension allows for `interface Ethernet1/1`
- 🧐 Why no leafref for name?



YANG

```
container interface {
    tailf:info "Physical interface configuration.";
    tailf:cli-add-mode;
    list Ethernet {
        tailf:info "List of ethernet interfaces.";
        tailf:cli-allow-join-with-key;
        key "name";
        leaf name {
            tailf:info "Ethernet interface name.";
            type string {
                pattern
                  '[0-9]+/[0-9]+(/[0-9]+)?';
            }
            must "count(/ciscolive/vpc-peer/member-interface[. = current()]) < 1" {
                error-message
                  "Interface cannot be used as a vPC peer link.";
            }
            must "count(../../port-channel/member-interface/name[. = current()]) < 1" {
                error-message
                  "Interface cannot be used as it is already a port-channel member.";
            }
            must "count(/ciscolive/vpc-peer/keepalive-interface[. = current()]) < 1" {
                error-message
                  "Interface cannot be used as it is already a vPC keepalive interface.";
            }
        }
```

The Power of Constraints

```
port-channel 66
   member-interface 1/35
    description-list [ "-> HX-FI-A_e1/35" "-> HX-FI-
A_e1/36" ]
   !
   description-list [ "-> DC1-HX-FI-A" "-> DC2-HX-FI-A" ]
   mode              trunk
   category          [ vm ]
```

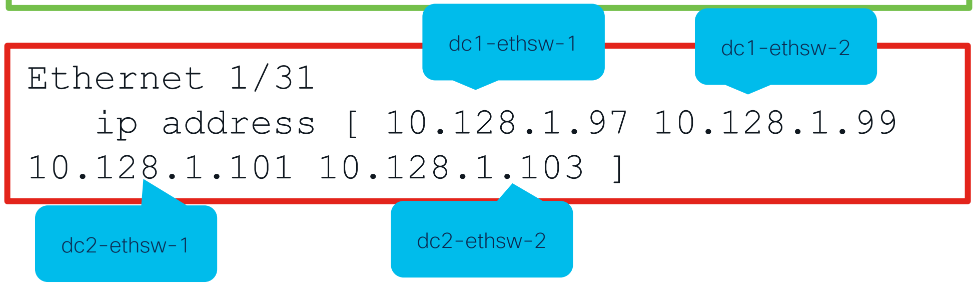
```
jclarke@ncs(config-ciscolive-Amsterdam/2023)# interface Ethernet1/35
jclarke@ncs(config-Ethernet-1/35)# ip address [ 1.1.1.1 1.1.1.2 1.1.1.3 1.1.1.4
]
jclarke@ncs(config-Ethernet-1/35)# commit
Aborted: 'ciscolive Amsterdam 2023 interface Ethernet 1/35 name' (value "1    ")
Interface cannot be used as it is already a port-channel member.
```

Ethernet Interface IPs

- An ethernet interface can be used for direct peering (i.e., to the network core)
- Note: `mode` here can be `edge` (DC edge), `access`, `trunk`, or `cross-dc-link`
- Therefore, each switch will have a different IP
- I chose a list approach using the switch's index

YANG

```
container ip {
    tailf:info "IPv4 commands for edge
interfaces.";
    leaf-list address {
        tailf:info "IPv4 address for
interface on each switch (used with
/31).";
        ordered-by user;
        type inet:ipv4-address;
        min-elements 4;
        max-elements 4;
        when '../../mode = "edge"';
    }
}
```



```
Ethernet 1/31
   ip address [ 10.128.1.97 10.128.1.99
10.128.1.101 10.128.1.103 ]
```

Port-channel Interfaces

- NX-OS makes you specify an interface is part of a port-channel
- I found it easier, especially for constraints, to have port-channels specify their members
- It’s your model. You have the flexibility

```
list member-interface {
    tailf:info "Member interfaces of this
port-channel.";
    key "name";
    leaf name {
        type string {
            pattern '[0-9]+/[0-9]+(/[0-
9]+)?';
        }
    }
    uses description-details;
    min-elements 1;
}
```

YANG

A Trade-Off Between Variability and Ease of Use

- Not everything needs to be a YANG element
- You know your service intent, so feel free to have more static parameters in the XML templates
- Or, put them in a separate variable Python file; or add them as `config false` nodes

XML

```
<ospf>
  <?if {$CATEGORY = "peer"}?>
  <authentication>
    <authentication-type>message-
digest</authentication-type>
    <key-chain>OSPF_KEY</key-chain>
  </authentication>
  <passive-interface>false</passive-interface>
  <?end?>
  <network>point-to-point</network>
</ospf>
<redirects>false</redirects>
<router>
  <ospf>
    <name>1</name>
    <area>0.0.0.0</area>
  </ospf>
</router>
```

Example of Static Parameters as config false

- Create a top-level OSPF area parameter
- Use `show ciscolive Amsterdam 2023 ospf-area` to display it
- Accessible inside Python as `service.ospf_area`
- Doesn’t clutter the `show running-config` output

```
leaf ospf-area {
    tailf:info "Static OSPF area ID";
    type inet:ipv4-address;
    default "0.0.0.0";
    config false;
}
```

Final Service Config

Almost...

- Final config at https://github.com/CiscoLearning/ciscolive-brkops-2040/blob/main/cl_2023.cfg
- Number of non-comment lines: **263**
- Number of resulting southbound config commands: **3467**

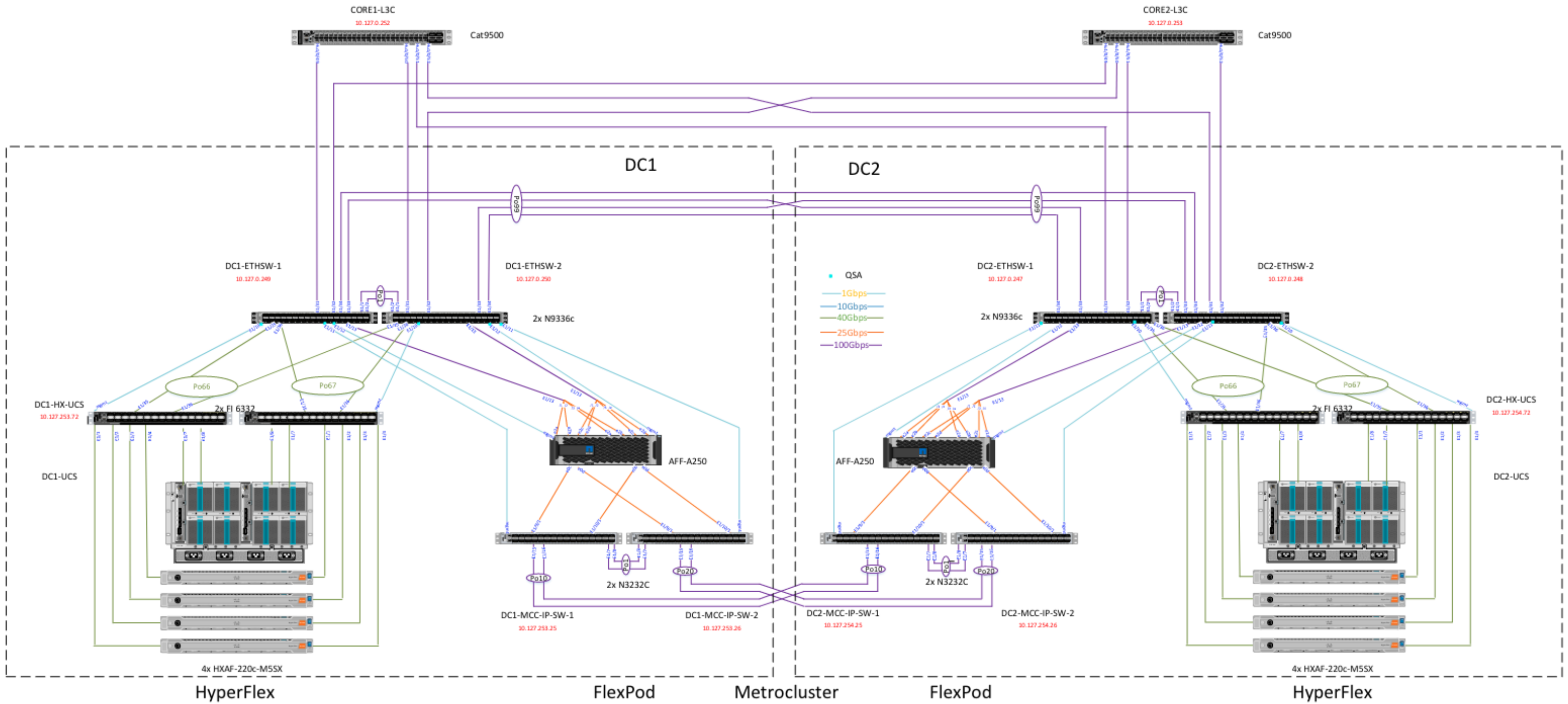
13x Config Compression!

Getting the Code

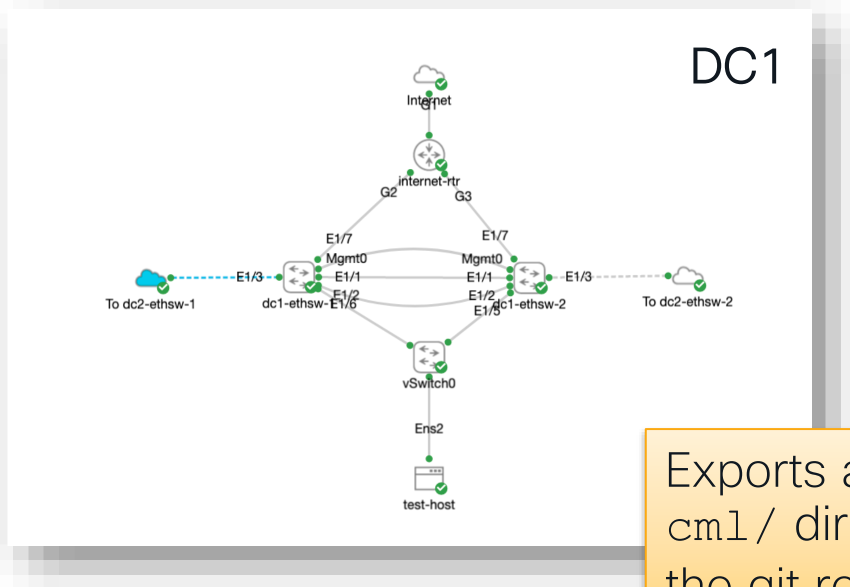
- Code is in GitHub at https://github.com/CiscoLearning/ciscolive-brkops-2040
- YANG module is in `src/yang/`
- XML templates are in `templates/`
- Python code is in `python/ciscolive/`

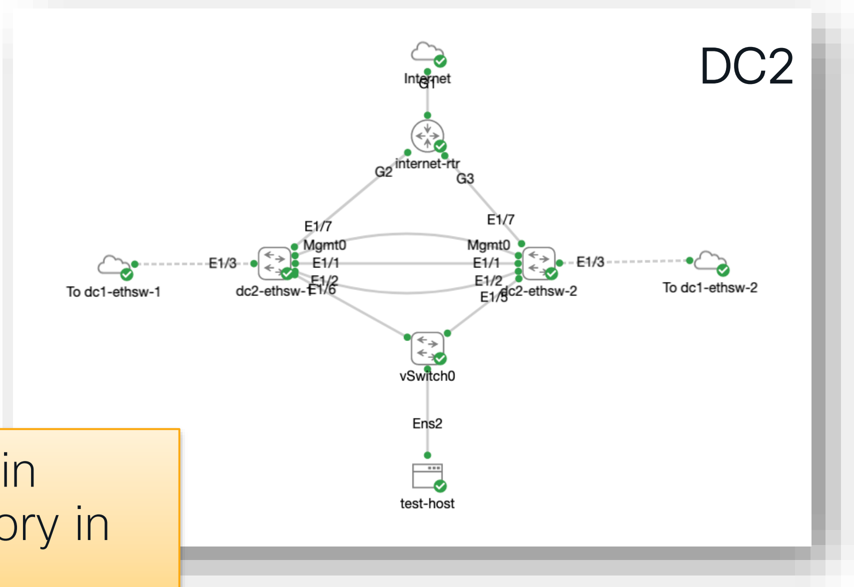
Testing Things Out

Our Actual Network



Our CML Test Network

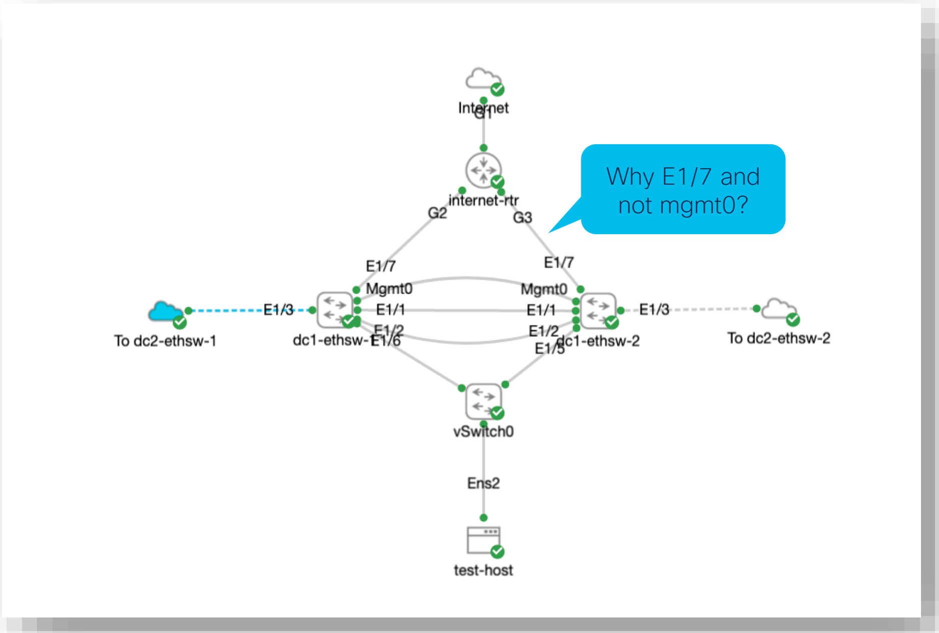




Exports are in `cml/` directory in the git repo

Connection Woes

- The current DC layout uses mgmt0 for the vPC keepalive, but connected back-to-back
- This is old technical debt that has not yet been removed
- It required more initial config on the switch and `re-deploy reconcile` to assume ownership of config by the service



Why E1/7 and not mgmt0?

Some Q&A

1. Why CML?
2. Why use two labs/topologies?
3. What about UCSM and vCenter?

1. While NSO Netsim is nice for strict model testing, CML let me confirm ultimate functionality.
2. Using two labs was a bit cleaner, and easier to iterate as the model expanded.
3. The networking pieces were the biggest lift. While I could have added the UCS simulator or a test vCenter, it wasn't as critical.

Testing Tips

- Commit frequently; use `un-deploy` if you want to back everything out
- As you iterate, use `re-deploy` to resync the network with the model
- Use `outformat` to aid in template development and understand your southbound changes
- The `dry-run` argument is your friend!

```
admin@ncs(config-ipv6)# commit dry-run outformat native
…
        data vlan 143
              name "CL_Test"
             exit
             interface Vlan143
              no shutdown
              description CL_Test
              ip address 10.143.252.247/24
              no ip arp gratuitous hsrp duplicate
              ip ospf network point-to-point
              no ip redirects
              ip router ospf 1 area 0.0.0.0
              ip flow monitor CL-ipv4-nb input
              ipv6 address 2a11:d940:2:8ffc::f7/64
```

```
admin@ncs(config-ipv6)# commit dry-run outformat xml
…
        data <ciscolive xmlns="http://example.com/ciscolive">
               <location>Amsterdam</location>
               <year>2023</year>
               <vlan>
                 <id>143</id>
                 <name>CL_TEST</name>
                 <routed>true</routed>
                 <dhcp>true</dhcp>
                 <ip>
                   <prefix>10.143.252.0/24</prefix>
                 </ip>
                 <ipv6>
                   <prefix>2a11:d940:2:8ffc::/64</prefix>
```

Looking Forward to 2024

Things I'd Do Differently

While the NSO solution worked quite well, there's always room for improvement. Remember, it's about iteration. Don't let perfect be the enemy of good.

Model

Use `config false`
Fold static data into the model as `config false` nodes

Compute IP[v6] prefixes
Prefixes are based on VLAN, so they need not be specified explicitly

Network

Dedicated out-of-band network
Will be much easier to provision if the mgmt0 interfaces are reachable by default

Integration

Incorporate source of truth more
We use NetBox for inventory management and IPAM. Use it for the authoritative list of VLANs so they only need to be added once

ACL Management
ACLs come from "The Tool". Either integrate with this as with NetBox or consider modeling them in NSO for the DC

What To Learn More?

- Come to our panel discussion on Friday, **PNLNMS-1035**
- Each part of the CiscoLive network will be presented
- We talk architecture, lessons learned and stats
- We know it will have been a long week; we make this talk fun!

CERTIFICATE

awarded to the

Cisco NOC Team

The most complex network of 750 APs & 500 switches
The management of 25K devices for 17K users
The longest cable rolled out (40km!)
A team of 60 world class engineers on-site
The least hours of sleep

OFFICIALLY AWESOME

What Will You Automate?

We have all come here to learn. What will you take back?

Brought to you by Cisco live!

Complete your Session Survey

- Please complete your session survey after each session. Your feedback is important.
- Complete a minimum of 4 session surveys and the Overall Conference survey (open from Thursday) to receive your Cisco Live t-shirt.

- All surveys can be taken in the Cisco Events Mobile App or by logging in to the Session Catalog and clicking the "Attendee Dashboard” at https://www.ciscolive.com/emea/learn/sessions/session-catalog.html

Continue Your Education

- Visit the Cisco Showcase for related demos.
- Book your one-on-one Meet the Engineer meeting.
- Attend any of the related sessions at the DevNet, Capture the Flag, and Walk-in Labs zones.
- Visit the On-Demand Library for more sessions at ciscolive.com/on-demand.

The bridge to possible

Thank you

cisco Live!