You make **possible**

Migration to Next-Gen Network with Automation First Approach

Akshaya Kumar, CX Consulting Engineer
@aks4talk

DEVNET-1047

Barcelona | January 27-31, 2020

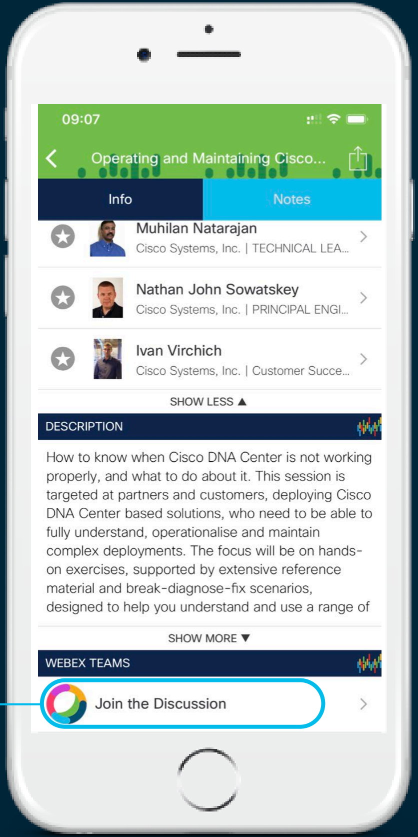
Cisco Webex Teams

Questions?

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

How

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



Muhilan Natarajan
Cisco Systems, Inc. | TECHNICAL LEA...

Nathan John Sowatskey
Cisco Systems, Inc. | PRINCIPAL ENGI...

Ivan Virchich
Cisco Systems, Inc. | Customer Succe...

DESCRIPTION

How to know when Cisco DNA Center is not working properly, and what to do about it. This session is targeted at partners and customers, deploying Cisco DNA Center based solutions, who need to be able to fully understand, operationalise and maintain complex deployments. The focus will be on hands-on exercises, supported by extensive reference material and break-diagnose-fix scenarios, designed to help you understand and use a range of

Agenda

- Introduction
- 3 Key Components –what's changing
- Automating Migration Activity
- Orchestration
- Conclusion

Introduction

Do you recall any of these situations

Let’s migrate in CLI mode first and then enable Programmability

Let me first deploy in safe mode as is and then further transition to Software Defined state

Large Aircraft : Longer Runway to fly::

Large Network : Migration planning

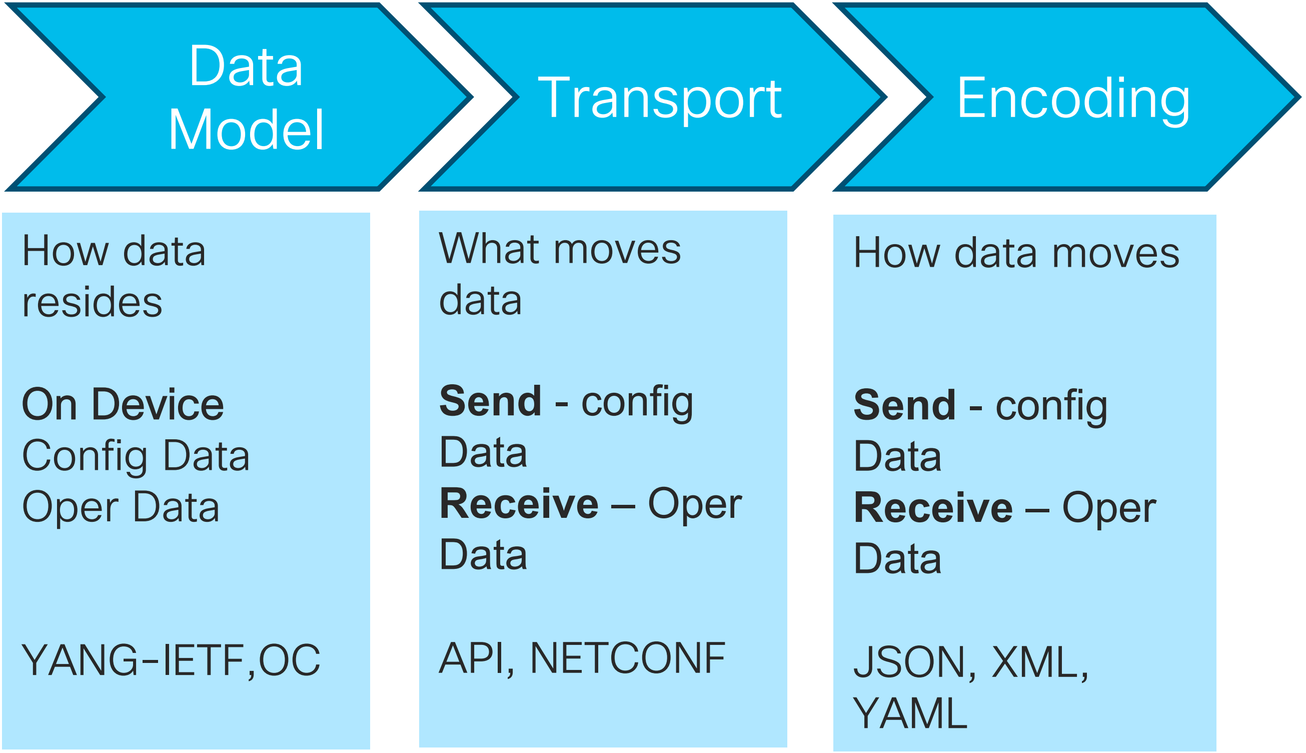
Network Operation
vs
Network Migration

- Operational tools are used for Day to Day Network Management
- Migration activity has unique requirements
- Reusing management tools for Migration sometimes limits what can be achieved
- Return on Investment seems less when features are not fully utilized

Key Components in Network Programming

...recap what is changing

3 Components of Programming

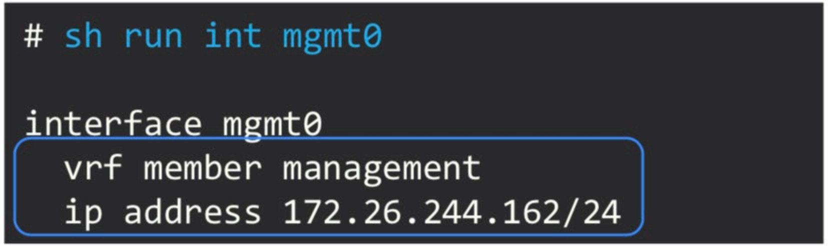


Data Model	Transport	Encoding
How data resides	What moves data	How data moves
On Device Config Data Oper Data	**Send** - config Data **Receive** – Oper Data	**Send** - config Data **Receive** – Oper Data
YANG-IETF,OC	API, NETCONF	JSON, XML, YAML

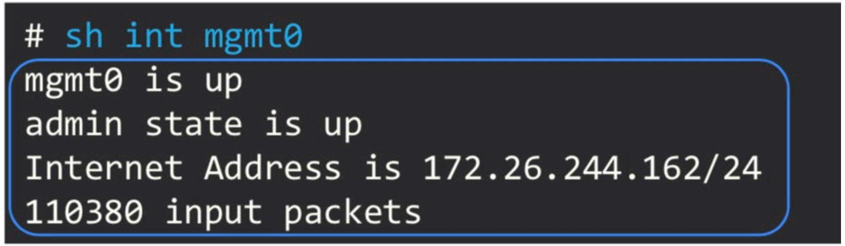
Device Data

Configuration data tells device what to do. It is data that we see in “show run”

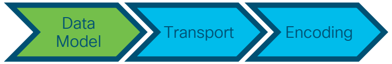
Operational data tells how devices is operating. All show commands other than show run



```
# sh run int mgmt0

interface mgmt0
  vrf member management
  ip address 172.26.244.162/24
```

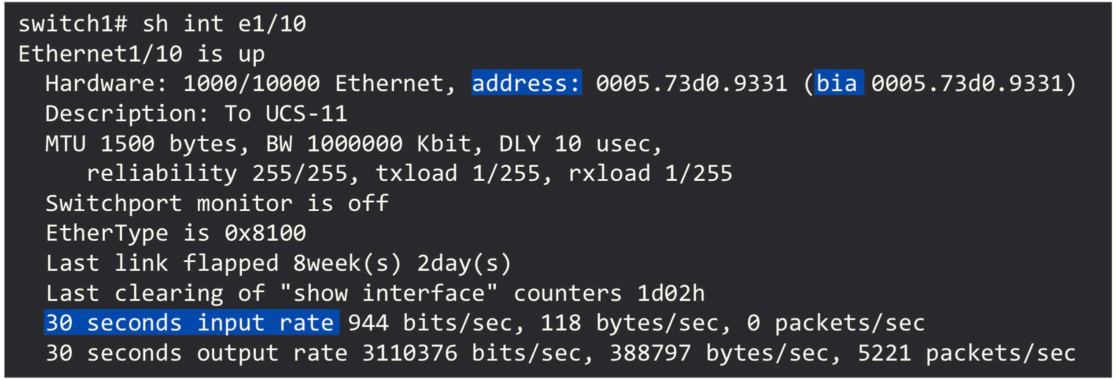


```
# sh int mgmt0
mgmt0 is up
admin state is up
Internet Address is 172.26.244.162/24
110380 input packets
```

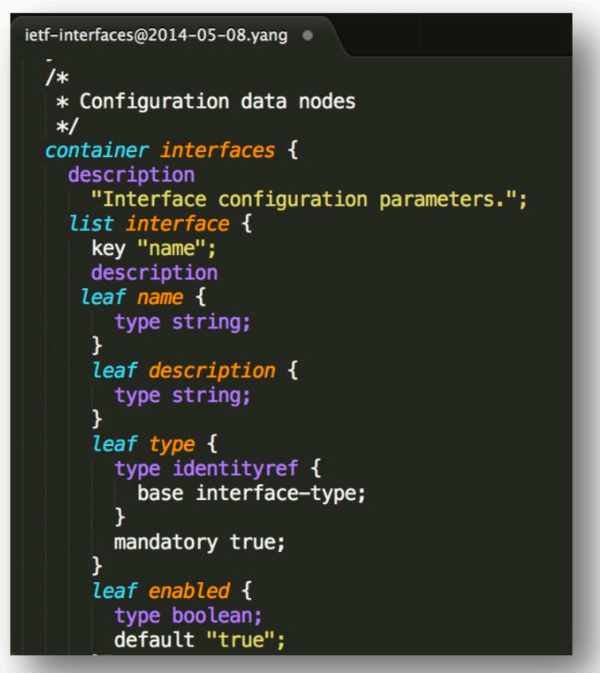


Unstructured vs Structured

Note inconsistent “key” format!



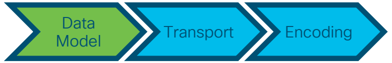
```
switch1# sh int e1/10
Ethernet1/10 is up
  Hardware: 1000/10000 Ethernet, address: 0005.73d0.9331 (bia 0005.73d0.9331)
  Description: To UCS-11
  MTU 1500 bytes, BW 1000000 Kbit, DLY 10 usec,
     reliability 255/255, txload 1/255, rxload 1/255
  Switchport monitor is off
  EtherType is 0x8100
  Last link flapped 8week(s) 2day(s)
  Last clearing of "show interface" counters 1d02h
  30 seconds input rate 944 bits/sec, 118 bytes/sec, 0 packets/sec
  30 seconds output rate 3110376 bits/sec, 388797 bytes/sec, 5221 packets/sec
```



ietf-interfaces@2014-05-08.yang

```
/*
 * Configuration data nodes
 */
container interfaces {
  description
    "Interface configuration parameters.";
  list interface {
    key "name";
    description
   leaf name {
      type string;
    }
    leaf description {
      type string;
    }
    leaf type {
      type identityref {
        base interface-type;
      }
      mandatory true;
    }
    leaf enabled {
      type boolean;
      default "true";
```

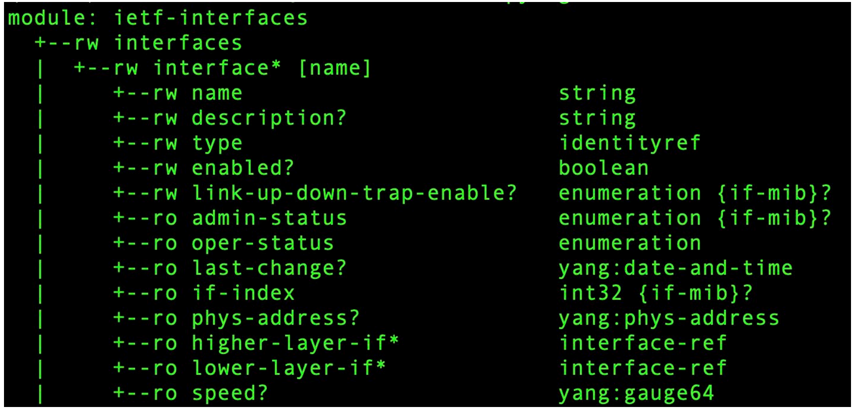
The recent development is Networking industry can be looked at as – Story of Data Model and Data format at Device & Network Level



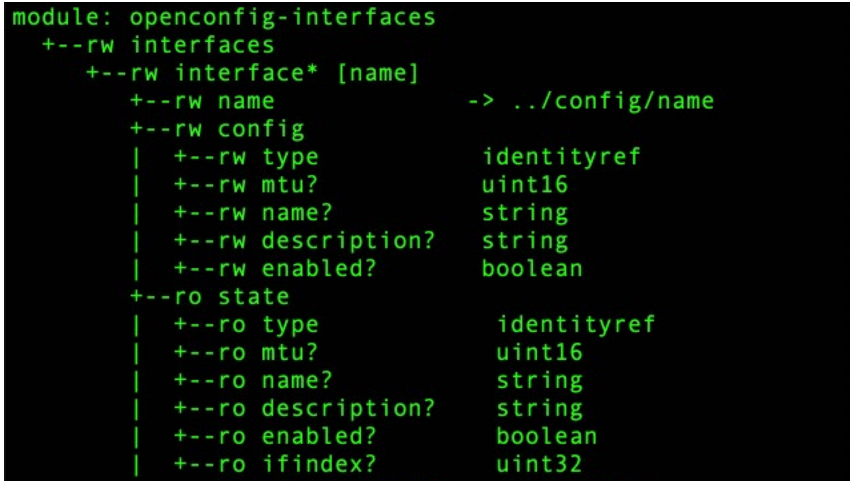
Data Model -- (1/2)

Device

- YANG Model
  - IETF
  - OpenConfig
  - Native



```
module: ietf-interfaces
  +--rw interfaces
  |  +--rw interface* [name]
  |     +--rw name                        string
  |     +--rw description?                string
  |     +--rw type                        identityref
  |     +--rw enabled?                    boolean
  |     +--rw link-up-down-trap-enable?   enumeration {if-mib}?
  |     +--ro admin-status                enumeration {if-mib}?
  |     +--ro oper-status                 enumeration
  |     +--ro last-change?                yang:date-and-time
  |     +--ro if-index                    int32 {if-mib}?
  |     +--ro phys-address?               yang:phys-address
  |     +--ro higher-layer-if*            interface-ref
  |     +--ro lower-layer-if*             interface-ref
  |     +--ro speed?                      yang:gauge64
```



```
module: openconfig-interfaces
  +--rw interfaces
     +--rw interface* [name]
        +--rw name                  -> ../config/name
        +--rw config
        |  +--rw type                identityref
        |  +--rw mtu?                uint16
        |  +--rw name?               string
        |  +--rw description?        string
        |  +--rw enabled?            boolean
        +--ro state
        |  +--ro type                 identityref
        |  +--ro mtu?                 uint16
        |  +--ro name?                string
        |  +--ro description?         string
        |  +--ro enabled?             boolean
        |  +--ro ifindex?             uint32
        |  +--ro admin-status
```

Data Model

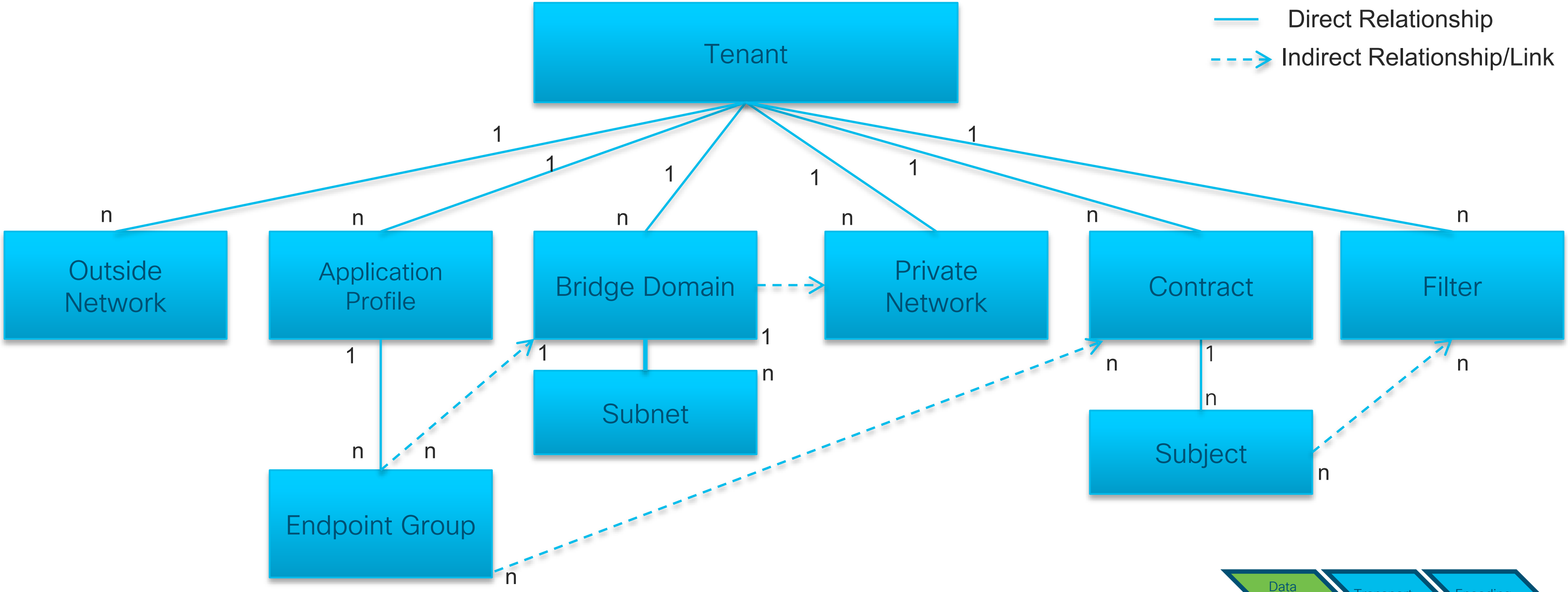
Transport

Encoding

Data Model – (2/2)

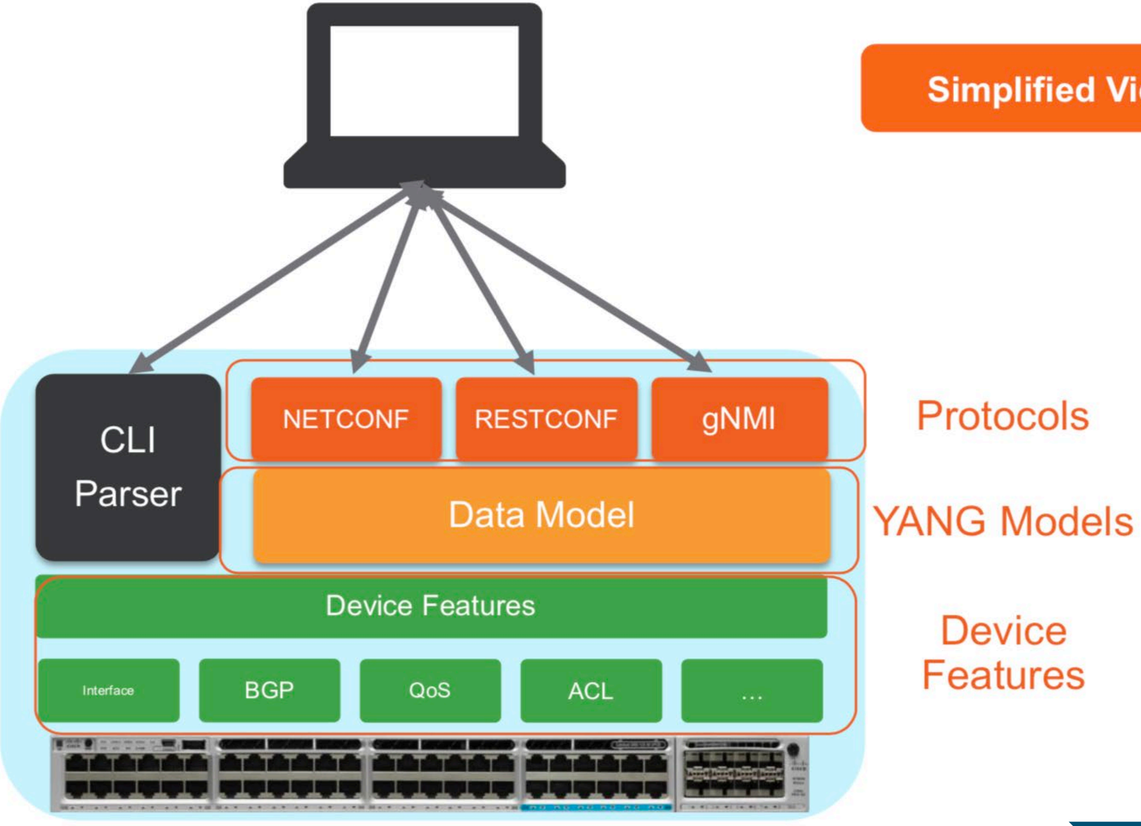
Network

1:n – one to many
n:n – many to many
Direct Relationship
Indirect Relationship/Link



ACI Data Model

Transport



Simplified View

Encoding : How data moves

We have this config information that must be sent between 2 Systems

```
IP
Source = 1.1.1.1
Destination = 2.2.2.2
```

What are the options to send it?

#serializing

```
<IP>
<source>1.1.1.1</source>
<destination>2.2.2.2</destination>
</IP>
```

XML : 73 bytes

```
{ IP :
          {source : 1.1.1.1},
          {destination: 2.2.2.2}
}
```

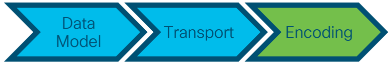
JSON : 58 bytes

```
IP:
 source:1.1.1.1
 destination:2.2.2.2
```

YAML : 45 bytes

```
00000001000000010000000100000001
00000010000000100000001000000010
```

Protobuf : 8 bytes



Demo : Device and Network Programmability

Automating Migration Activity

Network Migration Activities

Automation requirements at every step in Migration

- Discover current Network Services running, Operation state
- Plan, Prepare new Target Architecture
- Build Network Level or Device Level Low Level Design
- Deploy new Services/ Solutions
- Validate Service migration success

Automation is more focused in Deployment phase for scale it

Discover Phase

- Migration to a Programmable network starts from the way we start looking at the existing network
- For instance the Typical configuration of devices and Operational data in a Standard Data format JSON, XML, YAML etc
- Data Format also helps in Service Discovery - Several days to few hours with simple scripts
- Its possible to Capture Network attributes in standard format irrespective of deployment mode – traditional CLI or software-defined

Configuration & operational data in JSON/YAML

- Cisco NSO
- Tools from Devnet Community

(sample Discovery process in NSO)

```
!
interface Loopback0
 description **** Loopback0****
 ip address 1.1.1.1 255.255.255.255
!
```

show running-config devices device ios-device1 config | display json | save full_config2.json

```
<Snippet>

      "tailf-ned-cisco-ios:interface": {

                  "Loopback": [ { "name": "0",

                                          "description": "**** Loopback0****",

                                        "ip": { "address": {

                                                   "primary": { "address": "1.1.1.1",

                                                                   "mask": "255.255.255.255" }
```

Demo : Discover current solution

Plan & Design-Target Configuration

- Low Level design – captures what features needs to be enabled to implement the solution
- LLD must be accompanied by high level Templating language
- Templates can be translated to specific device configuration during implementation

Device implementation

CLI – Jinja2 and Unique Device data in YAML/ JSON Format

Device Level API – IOS XE, XR, NXOS

Controller implementation

Network Level API

NSO Service Templates

Config Templates

- Render configuration for multiple devices with templates

- Jinja template + YAML
- Render using Python, Ansible
  Optional: service templates in NSO

Jinja Configuration Template

```
hostname {{ config.name }}

interface Loopback0
ip address  {{ip_address}} 255.255.255.255

{% for vlan_id, vlan_name in config.vlans.items() %}
vlan {{ vlan_id }}
name {{ vlan_name }}
{% endfor %}
```

YAML Input templates

```
name: R1
vlans:
  10: Users
  20: Voice
ospf:
  - network: 10.0.1.0 0.0.0.255
    area: 0
  - network: 10.0.2.0 0.0.0.255
    area: 2
interfaces:
  - name: Gig0/0/0
    ip: 192.168.1.1
    mask: 255.255.255.0
```

Demo : Plan and Design

Templates

Orchestration

..building pipelines

Constructing Customized Workflows

Using high level orchestrator such as Ansible or NSO we can prepare workflows for Migration tasks.

Using standard Data structure used between Discover, Planning and Implementation steps helps flexible changes on various stages

- Service Discovery
- Pre and Post migration checks
- Prepare Target configuration & implementation

Service Discovery Reporting

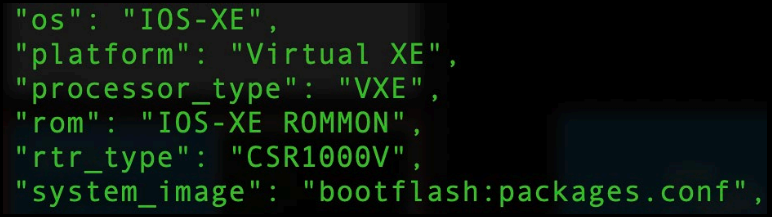
Building Service Discovery Pipeline to: - -
- Understand existing deployment
- Clients & Application Dependency mapping.
- Use network information in Standard format for reporting

Tools used:
Ansible orchestrator
Cisco pyATS – Parsegen
Python

Sample Ansible Playbook +cisco pyATS + Genie /ParseGenie

```
  tasks:
   - name: Read in parse_genie role
     include_role:
       name: clay584.parse_genie
   - name: Debug Genie Filter
     debug:
       msg: "{{ show_version_output | parse_genie(command='show version', os='iosxe') }}"
     delegate_to: localhost
```

Cisco pyATS + Genie



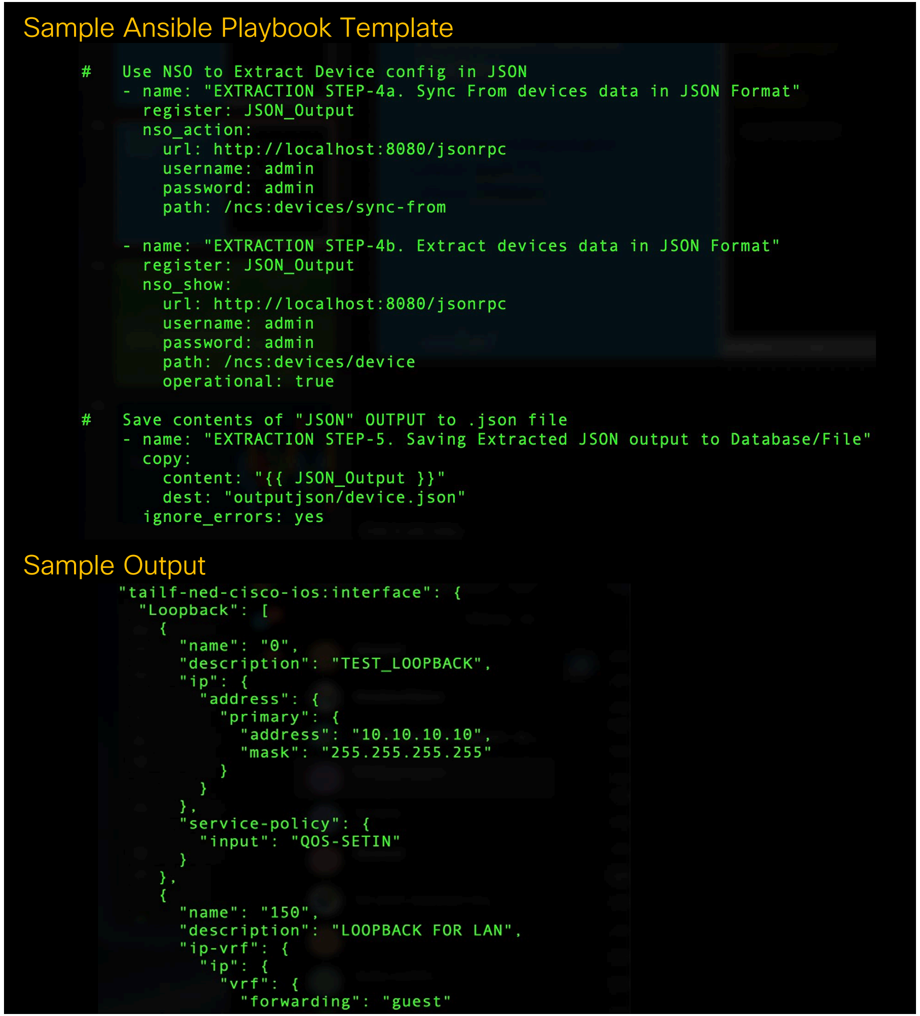
```
"os": "IOS-XE",
"platform": "Virtual XE",
"processor_type": "VXE",
"rom": "IOS-XE ROMMON",
"rtr_type": "CSR1000V",
"system_image": "bootflash:packages.conf",
```

Discover Device data + Implement

Building Pipeline to convert Configuration & Implement:
- Capture existing Configuration data
- Update attributes in standard format (JSON/YAML)
- Push configuration with Templates

Tools

Ansible orchestrator
NSO packages within Ansible
Jinja Templates



Sample Ansible Playbook Template

```
#   Use NSO to Extract Device config in JSON
    - name: "EXTRACTION STEP-4a. Sync From devices data in JSON Format"
      register: JSON_Output
      nso_action:
        url: http://localhost:8080/jsonrpc
        username: admin
        password: admin
        path: /ncs:devices/sync-from

    - name: "EXTRACTION STEP-4b. Extract devices data in JSON Format"
      register: JSON_Output
      nso_show:
        url: http://localhost:8080/jsonrpc
        username: admin
        password: admin
        path: /ncs:devices/device
        operational: true

#   Save contents of "JSON" OUTPUT to .json file
    - name: "EXTRACTION STEP-5. Saving Extracted JSON output to Database/File"
      copy:
        content: "{{ JSON_Output }}"
        dest: "outputjson/device.json"
      ignore_errors: yes
```

Sample Output

```
"tailf-ned-cisco-ios:interface": {
  "Loopback": [
    {
      "name": "0",
      "description": "TEST_LOOPBACK",
      "ip": {
        "address": {
          "primary": {
            "address": "10.10.10.10",
            "mask": "255.255.255.255"
          }
        }
      },
      "service-policy": {
        "input": "QOS-SETIN"
      }
    },
    {
      "name": "150",
      "description": "LOOPBACK FOR LAN",
      "ip-vrf": {
        "ip": {
          "vrf": {
            "forwarding": "guest"
```

Migration Validation

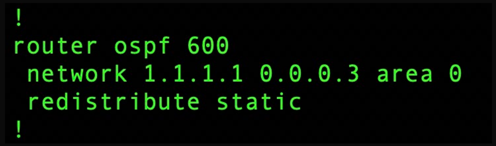
Building Pre- Post check Pipeline to identify successful migration. The Old network in Traditional CLI & new network via API

Tools

Ansible orchestrator
Python Scripts + custom data model
Open Python libraries

Thanks to Cisco Devnet Community & also following projects - Netmiko, Cisco pyATS, Parsegenie, ciscoconfparse, & other inspiring projects in

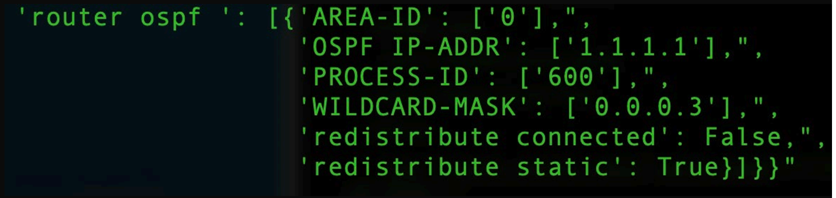
#Sample Config



```
!
router ospf 600
 network 1.1.1.1 0.0.0.3 area 0
 redistribute static
!
```

#Python Parser + Ansible (Demo)

#Output - Custom Data model + JSON



```
 'router ospf ': [{'AREA-ID': ['0'],",
                  'OSPF IP-ADDR': ['1.1.1.1'],",
                  'PROCESS-ID': ['600'],",
                  'WILDCARD-MASK': ['0.0.0.3'],",
                  'redistribute connected': False,",
                  'redistribute static': True}]}}"
```

#Optional Live Device - using ntc-templates & Netmiko

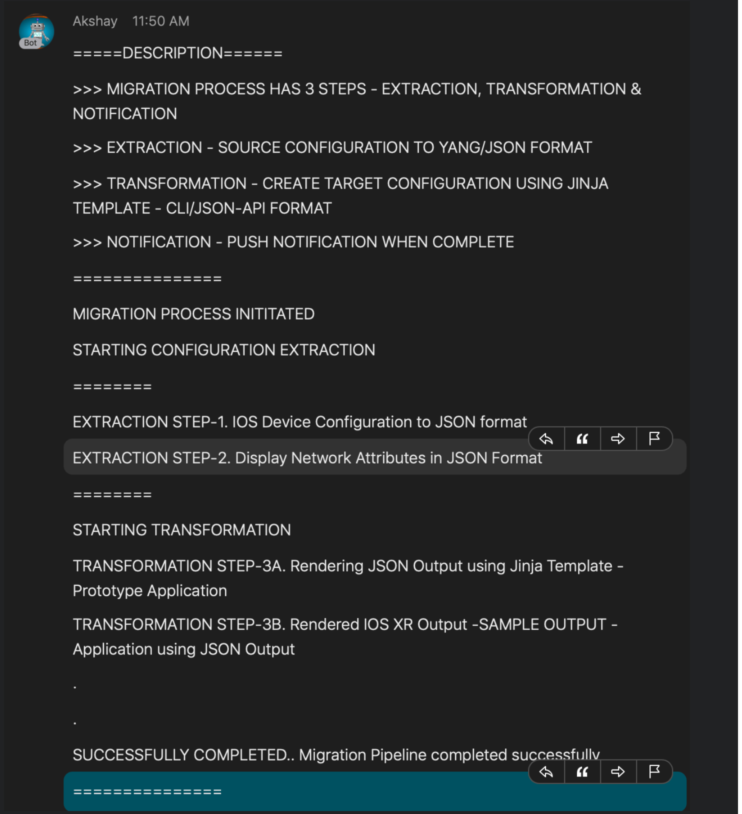
```
output = net_connect.send_command("show ip int brief", use_textfsm=True)

print(output)
```

Extra credits – Voice Activation+Cisco Webex

Pipeline with Cisco Webex Bot to update at every stage of Automation.

Tools
Ansible orchestrator
Cisco pyATS/ NSO/Python
Cisco Webex

Sample Webex Bot



Akshay 11:50 AM

=====DESCRIPTION======

>>> MIGRATION PROCESS HAS 3 STEPS - EXTRACTION, TRANSFORMATION & NOTIFICATION

>>> EXTRACTION - SOURCE CONFIGURATION TO YANG/JSON FORMAT

>>> TRANSFORMATION - CREATE TARGET CONFIGURATION USING JINJA TEMPLATE - CLI/JSON-API FORMAT

>>> NOTIFICATION - PUSH NOTIFICATION WHEN COMPLETE

===============

MIGRATION PROCESS INITITATED

STARTING CONFIGURATION EXTRACTION

========

EXTRACTION STEP-1. IOS Device Configuration to JSON format

EXTRACTION STEP-2. Display Network Attributes in JSON Format

========

STARTING TRANSFORMATION

TRANSFORMATION STEP-3A. Rendering JSON Output using Jinja Template - Prototype Application

TRANSFORMATION STEP-3B. Rendered IOS XR Output -SAMPLE OUTPUT - Application using JSON Output

.

.

SUCCESSFULLY COMPLETED.. Migration Pipeline completed successfully

===============

Demo : Creating workflow

Ansible, Cisco NSO, Cisco Webex, Open source tools

Complete your online session survey

- Please complete your session survey after each session. Your feedback is very important.
- Complete a minimum of 4 session surveys and the Overall Conference survey (starting on 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 Content Catalog on ciscolive.com/emea.

Cisco Live sessions will be available for viewing on demand after the event at ciscolive.com.

Continue your education

Demos in the Cisco Showcase

Walk-In Labs

Meet the Engineer 1:1 meetings

Related sessions

Thank you

You make **possible**