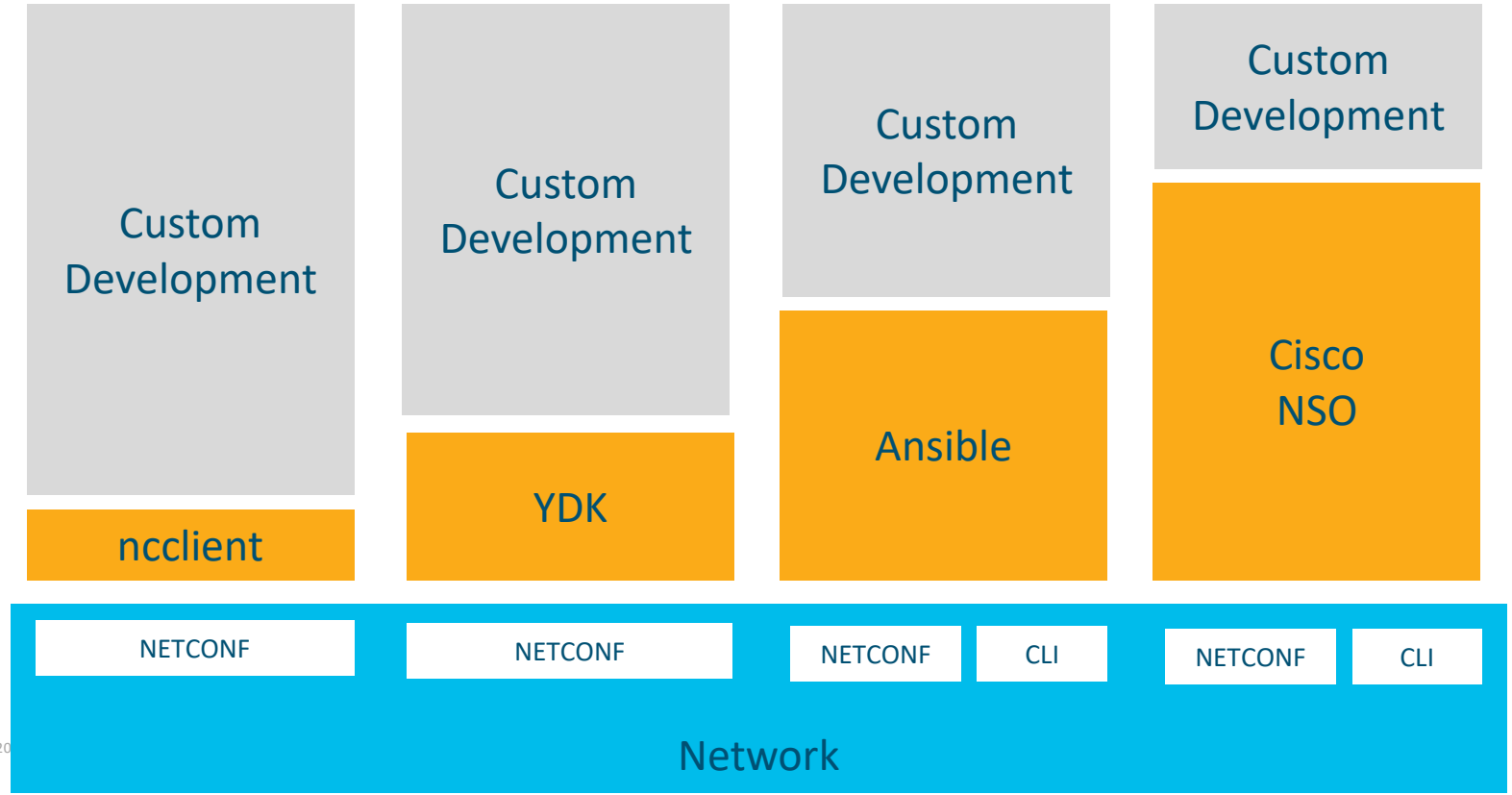
Four Options For Network Automation

https://github.com/djordjevulovic/4o4na

Djordje Vulovic
Systems Architect, CCIE #16582 Emeritus

Most Popular Technology Options for Network Automation



Use Case for Comparing Network Automation Options

Use Case
Description

- Automate following configuration steps:
  - Create new subinterface w/ IP address
  - Create new ACL and assign it to interface
  - Configure HSRP on interface
  - Add new network to OSPF area 1
- Additional automation:
  - Number of the new subinterface must be automatically derived
  - IP addresses (interface and the HSRP virtual) must be automatically derived from the IP prefix

Use Case
Sample Input and Output

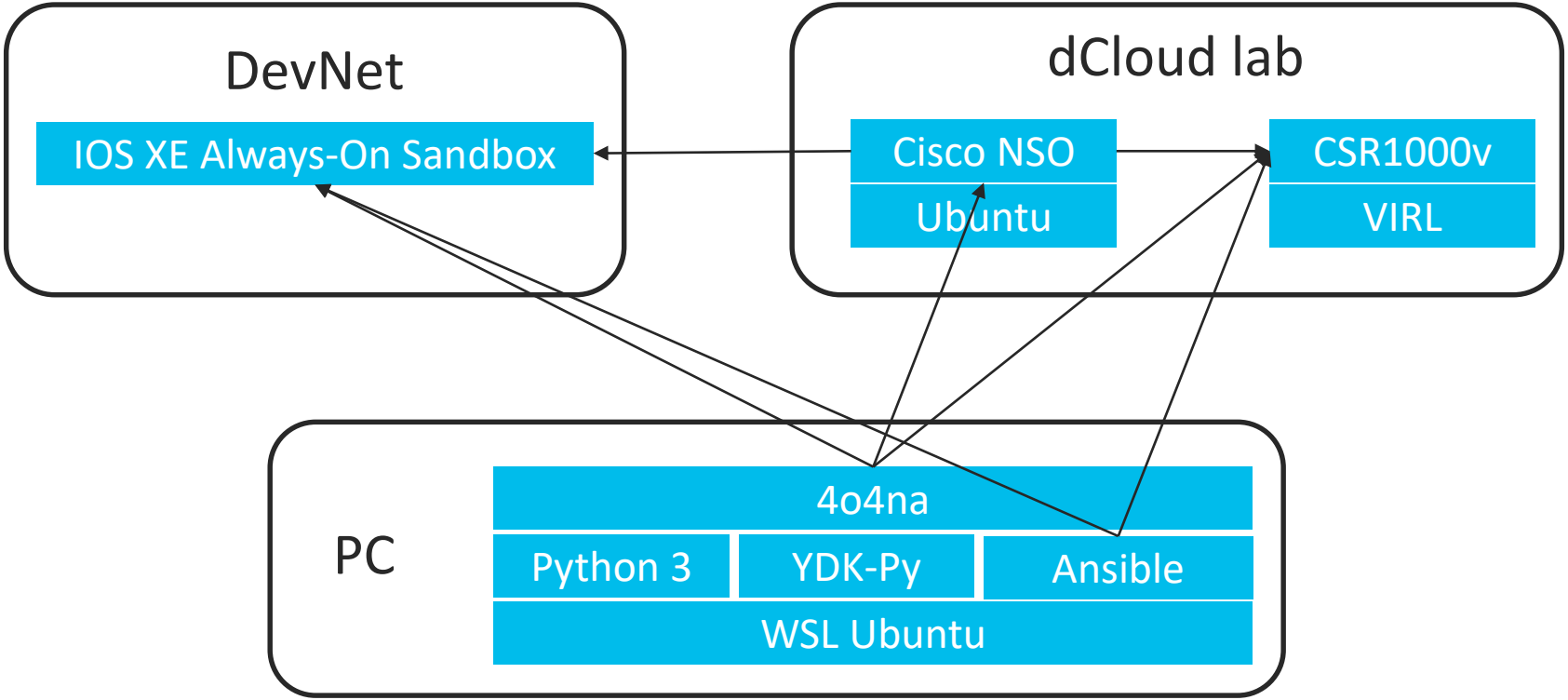
```
OPTION1
GigabitEthernet 3
192.168.1.0/24
```

```
ip access-list standard SAMPLE-ACL-in
 permit 192.168.1.0 0.0.0.255

interface GigabitEthernet3.101
 description SAMPLE
 encapsulation dot1Q 101
 ip address 192.168.1.2 255.255.255.0
 ip access-group SAMPLE-ACL-in in
 standby 1 ip 192.168.1.1
 standby 1 priority 120

router ospf 1
 network 192.168.1.0 0.0.0.255 area 1
```

Use Case
Example of Execution Environment



Use Case
Device Inventory

```
- hostname: dcloud
   ip: 198.18.1.33
   username: cisco
   password: cisco
   netconf_port: 830
   cli_port: 22
   proxy_ip: 198.18.1.30
   proxy_username: admin
   proxy_password: admin
   proxy_port: 8080
   proxy_hostname: ce2
```

```
- hostname: devnet_ios_xe
   ip: 64.103.37.51
   username: developer
   password: C1sco12345
   netconf_port: 10000
   cli_port: 8181
   proxy_ip: 198.18.1.30
   proxy_username: admin
   proxy_password: admin
   proxy_port: 8080
   proxy_hostname: devnet_ios_xe
```

Use Case
Starting Point (DevNet IOS XE Sandbox)

```
csr1000v#show ip access-lists
csr1000v#show ip interface brief
Interface              IP-Address      OK? Method Status                Protocol
GigabitEthernet1       10.10.20.48     YES NVRAM  up                    up
GigabitEthernet2       unassigned      YES NVRAM  administratively down down
GigabitEthernet3       unassigned      YES NVRAM  administratively down down
```

Option 1: Python + lxml + ncclient + NETCONF

XML Configuration in XE Native Model
Access-list (Example)

```
<native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
  <access-list>
    <standard xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-acl">
      <name>OPTION1-ACL-in</name>
        <access-list-seq-rule>
          <sequence>10</sequence>
          <permit>
             <std-ace>
                <ipv4-prefix>192.168.1.0</ipv4-prefix>
                <mask>0.0.0.255</mask>
              </std-ace>
          </permit>
        </access-list-seq-rule>
    </standard>
  </access-list>
</native>
```


https://github.com/YangModels/yang/blob/master/vendor/cisco/xe/1651/Cisco-IOS-XE-acl.yang

XML Configuration in XE Native Model
Subinterface (Example)

```
<native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
  <interface>
    <GigabitEthernet>
      <name>3.101</name>
      <description>OPTION1</description>
      <encapsulation><dot1Q><vlan-id>101</vlan-id></dot1Q></encapsulation>
      <ip>
        <access-group>
          <in><acl><acl-name>OPTION1-ACL-in</acl-name><in/></acl></in>
        </access-group>
        <address><primary>
          <address>192.168.1.2</address>
          <mask>255.255.255.0</mask></primary>
        </address>
      </ip>
      <standby><standby-list>
        <group-number>1</group-number>
        <ip><address>33.33.0.88</address></ip>
        <priority>120</priority>
      </standby-list></standby>
    </GigabitEthernet>
  </interface>
</native>
```

XML Configuration in XE Native Model
OSPF Network (Example)

```
<native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
  <router>
    <ospf xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-ospf">
      <id>1</id>
        <network>
          <ip>192.168.1.0</ip>
          <mask>0.0.0.255</mask>
           <area>1</area>
        </network>
     </ospf>
  </router>
</native>
```

https://github.com/YangModels/yang/blob/master/vendor/cisco/xe/1651/Cisco-IOS-XE-ospf.yang

Python Code
Building ACL Configuration

```
def create_acl(native_e, arg_svc_name, arg_prefix):

    netaddr_prefix = netaddr.IPNetwork(arg_prefix)

    ip_e = etree.SubElement(native_e, "ip")
    access_list_e = etree.SubElement(ip_e, "access-list")
    standard_e = etree.SubElement(access_list_e, "standard",nsmap={None:
'http://cisco.com/ns/yang/Cisco-IOS-XE-acl'})
    etree.SubElement(standard_e,"name").text= arg_svc_name + "-ACL-in"
    access_list_seq_rule_e = etree.SubElement(standard_e, "access-list-seq-rule")
    etree.SubElement(access_list_seq_rule_e,"sequence").text= '10'
    permit_e = etree.SubElement(access_list_seq_rule_e, "permit")
    std_ace_e = etree.SubElement(permit_e, "std-ace")
    etree.SubElement(std_ace_e,"ipv4-prefix").text= str(netaddr_prefix.network)
    etree.SubElement(std_ace_e,"mask").text= str(netaddr_prefix.hostmask)1
```

Python Code
Finding Out the New Subinterface Number

```
def find_next_subif(arg_if_type, arg_if_num):
    filter = "<filter><native xmlns='http://cisco.com/ns/yang/Cisco-IOS-XE-
native'><interface><{}></{}></interface></native></filter>".format(
        arg_if_type, arg_if_type)

    cfg = get_config(filter)

    max_subif = 0

    for action, elem in etree.iterparse(BytesIO(bytes(cfg, 'utf-8'))):
        if elem.tag == '{http://cisco.com/ns/yang/Cisco-IOS-XE-native}name':
            if '.' in elem.text:
                if_parts = elem.text.split('.')
                if if_parts[0] == arg_if_num:
                    max_subif = int(if_parts[1])

    return max_subif + 1
```

Python Code
Building Subinterface Configuration

```
def create_subif(native_e, arg_svc_name, arg_if_type,
arg_if_num, arg_prefix):
    subif = str(find_next_subif(arg_if_type, arg_if_num))

    netaddr_prefix = netaddr.IPNetwork(arg_prefix)

    interface_e = etree.SubElement(native_e, "interface")
    gigabitethernet_e = etree.SubElement(interface_e,
"GigabitEthernet")
    etree.SubElement(gigabitethernet_e, "name").text =
arg_if_num + "." + subif
    etree.SubElement(gigabitethernet_e, "description").text
= arg_svc_name
    encapsulation_e = etree.SubElement(gigabitethernet_e,
"encapsulation")
    dot1q_e = etree.SubElement(encapsulation_e, "dot1Q")
    etree.SubElement(dot1q_e, "vlan-id").text = subif

    ip_e = etree.SubElement(gigabitethernet_e, "ip")

    access_group_e = etree.SubElement(ip_e, "access-group")
    in_e = etree.SubElement(access_group_e, "in")
    acl_e = etree.SubElement(in_e, "acl")
    etree.SubElement(acl_e, "acl-name").text = arg_svc_name
+ "-ACL-in"
    in2_e = etree.SubElement(acl_e, "in")
```

```
    address_e = etree.SubElement(ip_e, "address")
    primary_e = etree.SubElement(address_e, "primary")
    etree.SubElement(primary_e, "address").text =
str(list(netaddr_prefix)[2])
    etree.SubElement(primary_e, "mask").text =
str(netaddr_prefix.netmask)

    standby_e = etree.SubElement(gigabitethernet_e, "standby")
    standby_list_e = etree.SubElement(standby_e, "standby-list")
    etree.SubElement(standby_list_e, "group-number").text = '1'
    ip2_e = etree.SubElement(standby_list_e, "ip")
    etree.SubElement(ip2_e, "address").text =
str(list(netaddr_prefix)[1])
    etree.SubElement(standby_list_e, "priority").text = '120'

    return subif
```

Python Code
Building OSPF Network Configuration

```
def create_ospf_network(native_e, arg_prefix):

    netaddr_prefix = netaddr.IPNetwork(arg_prefix)

    router_e = etree.SubElement(native_e, "router")
    ospf_e = etree.SubElement(router_e, "ospf",nsmap={None: 'http://cisco.com/ns/yang/Cisco-
IOS-XE-ospf'})
    etree.SubElement(ospf_e,"id").text= '1'
    network_e = etree.SubElement(ospf_e, "network")
    etree.SubElement(network_e,"ip").text= str(netaddr_prefix.network)
    etree.SubElement(network_e,"mask").text= str(netaddr_prefix.hostmask)
    etree.SubElement(network_e,"area").text= '1'
```

Python Code
Service Configuration

```
def create_service(arg_svc_name, arg_if_type, arg_if_num, arg_prefix, arg_dryrun=False):
    config_e = etree.Element("config")
    native_e = etree.SubElement(config_e, "native", nsmap={None:
'http://cisco.com/ns/yang/Cisco-IOS-XE-native'})

    create_acl(native_e, arg_svc_name, arg_prefix)
    subif = create_subif(native_e, arg_svc_name, arg_if_type, arg_if_num, arg_prefix)
    create_ospf_network(native_e, arg_prefix)

    if arg_dryrun is False:
        if push_config(config_e) is True:
            return str(subif)
        else:
            return None
    else:
        xml_prettyprint(config_e)
        return subif
```

Option 1 Demo
Demo Service Instance

```
dvulovic@DVULOVIC-V2TQV:~/4o4na$ cat demosvc1.yml
---
parameters:
  service_name: OPTION1
  device: devnet_ios_xe
  interface_type: GigabitEthernet
  interface_number: 3
  ip_prefix: 192.168.1.0/24
```

Option 1 Demo

Run Script in “dry-run” Mode

```
dvulovic@DVULOVIC-V2TQV:~/4o4na$ ./create_service.py -s demosvc1.yml -o 1 -y
<config>
  <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
    <ip>
      <access-list>
        <standard xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-acl">
          <name>OPTION1-ACL-in</name>
          <access-list-seq-rule>
            <sequence>10</sequence>
            <permit>
              <std-ace>
                <ipv4-prefix>192.168.1.0</ipv4-prefix>
                <mask>0.0.0.255</mask>
              </std-ace>
            </permit>
          </access-list-seq-rule>
        </standard>
      </access-list>
…
 </native>
</config>
```

Option 1 Demo

Run Script and Check Results (DevNet IOS XE Sandbox)

```
dvulovic@DVULOVIC-V2TQV:~/4o4na$ ./create_service.py -s demosvc1.yml -o 1
Service created.
New subinterface GigabitEthernet3.1
```

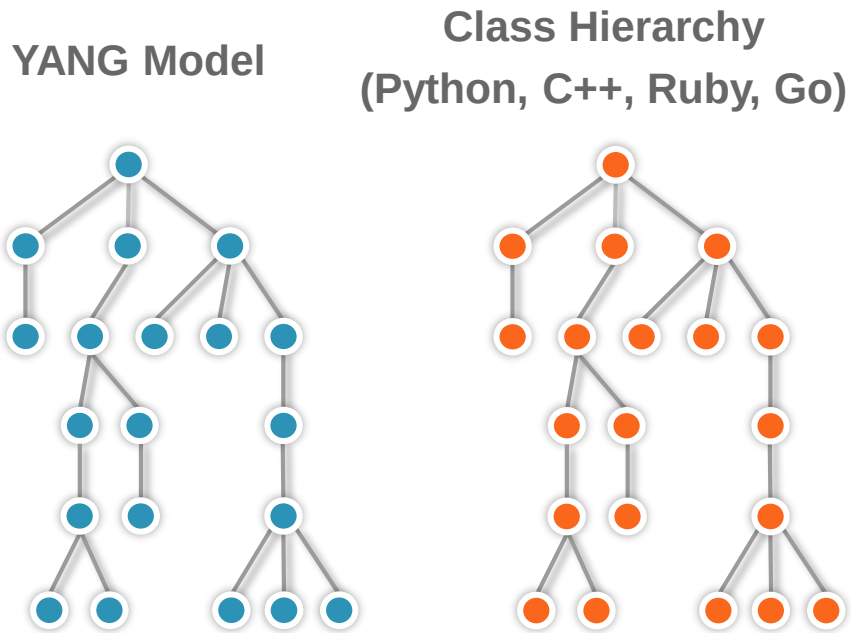
```
csr1000v#sh ip access-lists
Standard IP access list OPTION1-ACL-in
    10 permit 192.168.1.0, wildcard bits 0.0.0.255
csr1000v#sh runn int gi 3.1
Building configuration...

Current configuration : 210 bytes
!
interface GigabitEthernet3.1
 description OPTION1
 encapsulation dot1Q 1 native
 ip address 192.168.1.2 255.255.255.0
 ip access-group OPTION1-ACL-in in
 standby 1 ip 192.168.1.1
 standby 1 priority 120
end
```

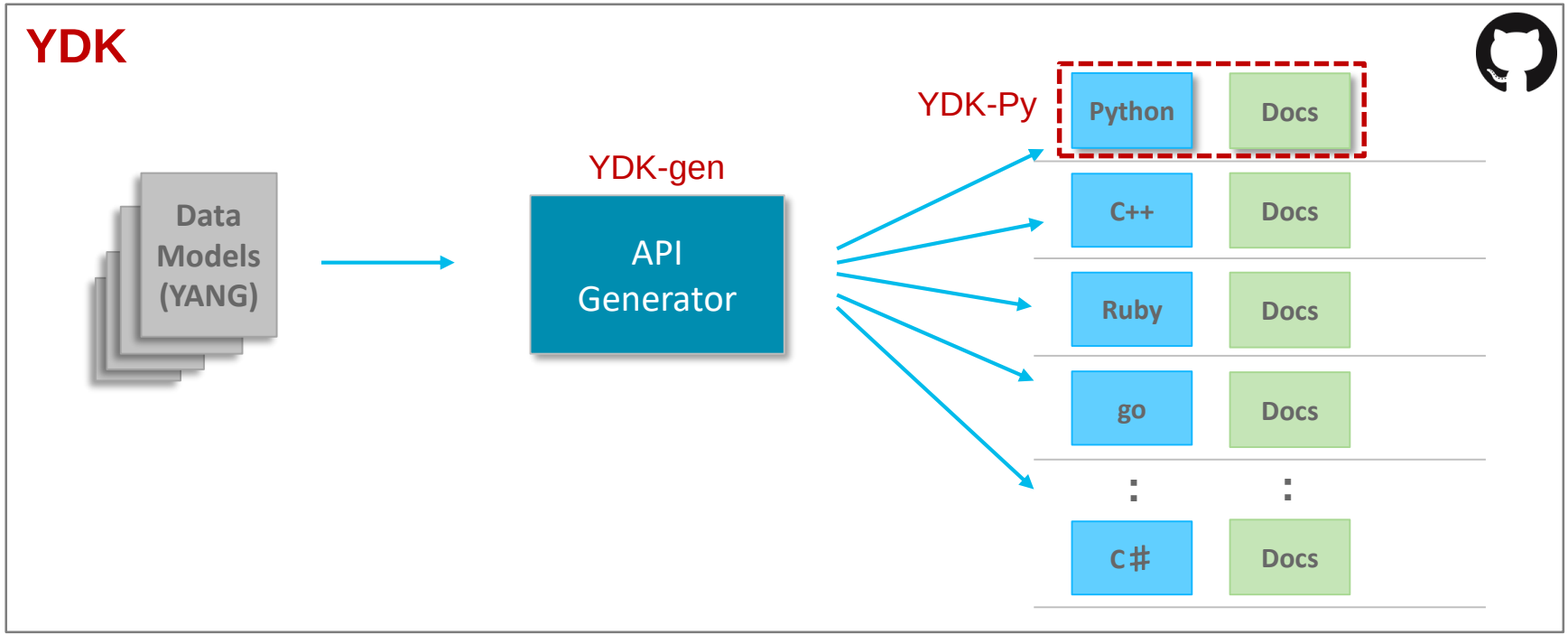
Option 2: Python + YDK-Py

Model-Driven APIs

- Simplify app development
- Abstract transport, encoding, modeling language
- API generated from YANG model
- One-to-one correspondence between model and class hierarchy
- Multi-language (Python, C++, Ruby, Go, etc.)



Generation of Model-Driven APIs Using YANG Development Kit (YDK)



Python Code
Building ACL Configuration

```
def create_acl(xe_native_config, arg_svc_name, arg_prefix):
    netaddr_prefix = netaddr.IPNetwork(arg_prefix)

    acl = xe_native_config.ip.access_list

    new_acl = acl.Standard()
    new_acl.name = arg_svc_name + "-ACL-in"

    new_rule = new_acl.AccessListSeqRule()
    new_rule.sequence = 10
    new_rule.permit.std_ace.ipv4_prefix = str(netaddr_prefix.network)
    new_rule.permit.std_ace.mask = str(netaddr_prefix.hostmask)

    new_acl.access_list_seq_rule.append(new_rule)
    acl.standard.append(new_acl)
```

Python Code
Finding Out the New Subinterface Number

```
def find_next_subif(xe_native_config, arg_if_type, arg_if_num):
    filter = xe_native_config.interface
    config = ydk_crud.read(ydk_provider, filter)

    max_subif = 0

    if_type_list = {"GigabitEthernet": config.gigabitethernet}

    for intf in if_type_list[arg_if_type]:
        if '.' in intf.name:
            if_parts = intf.name.split('.')
            if if_parts[0] == arg_if_num:
                max_subif = int(if_parts[1])

    return max_subif + 1
```

Python Code
Building Subinterface Configuration

```
def create_subif(xe_native_config, arg_svc_name, arg_if_type, arg_if_num, arg_prefix):
    ge_list = xe_native_config.interface.gigabitethernet

    new_if = xe_native_config.interface.GigabitEthernet()

    subif = find_next_subif(xe_native_config, arg_if_type, arg_if_num)

    netaddr_prefix = netaddr.IPNetwork(arg_prefix)

    new_if.name = arg_if_num + "." + str(subif)
    new_if.description = arg_svc_name
    new_if.encapsulation.dot1q.vlan_id = subif
    new_if.ip.address.primary.address = str(list(netaddr_prefix)[2])
    new_if.ip.address.primary.mask = str(netaddr_prefix.netmask)

    new_if.ip.access_group.in_.acl.acl_name = arg_svc_name + "-ACL-in"
    new_if.ip.access_group.in_.acl.in_ = ydk.types.Empty()

    new_standby_list = new_if.standby.StandbyList()
    new_standby_list.group_number = 1
    new_standby_list.ip = new_standby_list.Ip()
    new_standby_list.ip.address = str(list(netaddr_prefix)[1])
    new_standby_list.priority = 120
    new_if.standby.standby_list.append(new_standby_list)

    ge_list.append(new_if)

    return subif
```

Python Code
Building OSPF Network Configuration

```
def create_ospf_network(xe_native_config, arg_prefix):
    netaddr_prefix = netaddr.IPNetwork(arg_prefix)

    ospf_process = xe_native_config.router.Ospf()
    ospf_process.id = 1
    ospf_process_network = ospf_process.Network()
    ospf_process_network.ip = str(netaddr_prefix.network)
    ospf_process_network.mask = str(netaddr_prefix.hostmask)
    ospf_process_network.area = 1
    ospf_process.network.append(ospf_process_network)
    xe_native_config.router.ospf.append(ospf_process)
```

Python Code
Service Configuration

```
def create_service(arg_svc_name, arg_if_type, arg_if_num, arg_prefix, arg_dryrun=False):
    xe_native_config = xe_native.Native()

    create_acl(xe_native_config, arg_svc_name, arg_prefix)
    subif = create_subif(xe_native_config, arg_svc_name, arg_if_type, arg_if_num, arg_prefix)
    create_ospf_network(xe_native_config, arg_prefix)

    if arg_dryrun is False:
        ydk_crud.create(ydk_provider, xe_native_config)
        return str(subif)
    else:
        codec_service = CodecService()
        codec_provider = CodecServiceProvider(type='xml')

        print(codec_service.encode(codec_provider, xe_native_config))
        return str(subif)
```

Option 2 Demo
Demo Service Instance

```
dvulovic@DVULOVIC-V2TQV:~/4o4na$ cat demosvc2.yml
---
parameters:
  service_name: OPTION2
  device: devnet_ios_xe
  interface_type: GigabitEthernet
  interface_number: 3
  ip_prefix: 192.168.2.0/24
```

Option 2 Demo

Run Script in “dry-run” Mode

```
dvulovic@DVULOVIC-V2TQV:~/4o4na$ ./create_service.py -s demosvc2.yml -o 2 -y
<native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
  <ip>
    <access-list>
      <standard xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-acl">
        <name>OPTION2-ACL-in</name>
        <access-list-seq-rule>
          <sequence>10</sequence>
          <permit>
            <std-ace>
              <ipv4-prefix>192.168.2.0</ipv4-prefix>
              <mask>0.0.0.255</mask>
            </std-ace>
          </permit>
        </access-list-seq-rule>
      </standard>
    </access-list>
…
 </native>
</config>
```

Option 2 Demo

Run Script and Check Results (DevNet IOS XE Sandbox)

```
dvulovic@DVULOVIC-V2TQV:~/4o4na$ ./create_service.py -s demosvc2.yml -o 2
Service created.
New subinterface GigabitEthernet3.2
```

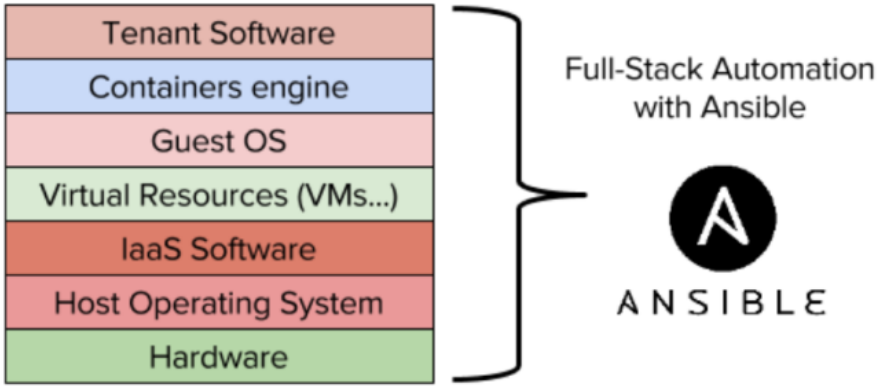
```
csr1000v#sh ip access-lists
Standard IP access list OPTION1-ACL-in
    10 permit 192.168.1.0, wildcard bits 0.0.0.255
Standard IP access list OPTION2-ACL-in
    10 permit 192.168.2.0, wildcard bits 0.0.0.255
csr1000v#sh runn int gi 3.2
Building configuration...

Current configuration : 203 bytes
!
interface GigabitEthernet3.2
 description OPTION2
 encapsulation dot1Q 2
 ip address 192.168.2.2 255.255.255.0
 ip access-group OPTION2-ACL-in in
 standby 1 ip 192.168.2.1
 standby 1 priority 120
end
```

Option 3: Ansible + Jinja2 + IOS Modules + CLI

Ansible

- Open source software that automates software provisioning, configuration management, and application deployment.
  - Red Hat offers support for Ansible with Red Hat Ansible Engine product
- Agentless architecture (no deamon required)



Managing Network Devices with Ansible

- Ansible delivers native support for Cisco IOS / IOS-XE / IOS-XR / NX-OS platforms:
  - Executing operational commands (e.g. “ios_command” module)
  - Executing config commands (e.g. “ios_config” module)
  - etc.
- Typical process for network devices:
  - Ansible connect to local server and runs module’s Python code
  - Module connects to device and executes commands

Ansible Playbook
Ansible Tasks for Finding New Subinterface Number (part 1)

```
- name: Get IOS facts
  ios_facts:
    gather_subset:
       - interfaces
    provider:
        host: "{{ device_ip }}"
        username: "{{ device_username }}"
        password: "{{ device_password }}"
        port: "{{ device_port }}"
  register: output1

- name: Create list of all interfaces
  set_fact:
     all_interfaces: "{{ output1.ansible_facts.ansible_net_interfaces | list }}"

- set_fact:
     subif_match_pattern: "{{ '^{{ interface_type }}{{ interface_number }}\\..*' }}"
```

Ansible Playbook
Ansible Tasks for Finding New Subinterface Number (Part 2)

```
- name: Create list of applicable subinterfaces
  set_fact:
     subifs: "{{ all_interfaces |  select('match', subif_match_pattern) | list }}"

- set_fact:
     subif_replace_pattern: "{{ '^{{ interface_type }}{{ interface_number }}\\.(?P<subif>[0-9]+)' }}"

- name: Get new subinterface number
  set_fact:
     subif: "{{ subif_nums | max | int + 1 }}"
  when:
    - subif_nums != []

- name: Set subinterface number to 1 if it is the first one in the interface
  set_fact:
     subif: "1"
  when:
    - subif_nums == []
```

Ansible Playbook
Configuration Template

```
ip access-list standard {{ service_name }}-ACL-in
 permit {{ ip_prefix | ipaddr('network') }} {{ ip_prefix | ipaddr('wildcard') }}

interface {{ interface_type }}{{ interface_number }}.{{ subif }}
 description {{ service_name }}
 encapsulation dot1Q {{ subif }}
 ip address {{ ip_prefix | ipaddr(2) | ipaddr('address') }} {{ ip_prefix |
ipaddr('netmask') }}
 ip access-group {{ service_name }}-ACL-in in
 standby 1 ip {{ ip_prefix | ipaddr(1) | ipaddr('address') }}
 standby 1 priority 120

router ospf 1
 network {{ ip_prefix | ipaddr('network') }} {{ ip_prefix | ipaddr('wildcard') }} area 1
```

Ansible Playbook
Ansible Tasks for Dry-Run

```
- name: "Service configuration (dry run)"
     template:
        src: option3.j2
        dest: "/tmp/dry-run-{{ service_name }}.cfg"
     when:
       dry_run | bool

   - name: Show dry-run configuration
     debug: var=item
     with_file:
       - "/tmp/dry-run-{{ service_name }}.cfg"
     when:
       dry_run and debug | bool"
```

Ansible Playbook
Ansible Task for Actual Configuration

```
- name: Service confguration (device)
    ios_config:
      provider:
          host: "{{ device_ip }}"
          username: "{{ device_username }}"
          password: "{{ device_password }}"
          port: "{{ device_port }}"
      src: option3.j2
    when:
      not dry_run | bool
```

Python Code
Running Ansible Playbook

```
def create_service(arg_svc_name, arg_if_type, arg_if_num, arg_prefix, arg_dryrun=False):
    global global_variable_manager

    playbook_path = 'option3.yml'

    if not os.path.exists(playbook_path):
        print("No playbook found!")
        return None

    global_variable_manager.extra_vars['service_name'] = arg_svc_name
    global_variable_manager.extra_vars['interface_type'] = arg_if_type
    global_variable_manager.extra_vars['interface_number'] = arg_if_num
    global_variable_manager.extra_vars['ip_prefix'] = arg_prefix
    global_variable_manager.extra_vars['dry_run'] = "true" if arg_dryrun is True else "false"

    context.CLIARGS = ImmutableDict(connection='local', forks=10, become=None, become_method=None, become_user=None,
                                    check=False, diff=False, syntax=False, start_at_task=None, verbosity=1)

    pbex = PlaybookExecutor(playbooks=[playbook_path],
                            variable_manager=global_variable_manager,
                            loader=global_loader,
                            inventory=global_inventory,
                            passwords={})

    results = pbex.run()
    print(results)
    return ''
```

Option 3 Demo
Demo Service Instance

```
dvulovic@DVULOVIC-V2TQV:~/4o4na$ cat demosvc3.yml
---
parameters:
  service_name: OPTION3
  device: devnet_ios_xe
  interface_type: GigabitEthernet
  interface_number: 3
  ip_prefix: 192.168.3.0/24
```

Option 3 Demo

Run Script in “dry-run” Mode

```
dvulovic@DVULOVIC-V2TQV:~/4o4na$ ./create_service.py -s demosvc3.yml -o 3 -y

PLAY [localhost]
*****************************************************************************************************************
*********************************************
…

TASK [Show dry-run configuration]
*****************************************************************************************************************
*****************************
skipping: [localhost] => (item=ip access-list standard OPTION3-ACL-in
 permit 192.168.3.0 0.0.0.255

interface GigabitEthernet3.3
 description OPTION3
 encapsulation dot1Q 3
 ip address 192.168.3.2 255.255.255.0
 ip access-group OPTION3-ACL-in in
 standby 1 ip 192.168.3.1
 standby 1 priority 120

router ospf 1
 network 192.168.3.0 0.0.0.255 area 1)
skipping: [localhost]
…
```

Option 3 Demo
Run Script

```
dvulovic@DVULOVIC-V2TQV:~/4o4na$ ./create_service.py -s demosvc3.yml -o 3

PLAY [localhost]
*********************************************************************************************************************
*********************************************
…

TASK [Service confguration (device)]
*********************************************************************************************************************
****************************
changed: [localhost]

PLAY RECAP
*********************************************************************************************************************
**************************************************
localhost                  : ok=8    changed=1    unreachable=0    failed=0    skipped=9    rescued=0
ignored=0
```

Option 3 Demo
Check Results (DevNet IOS XE Sandbox)

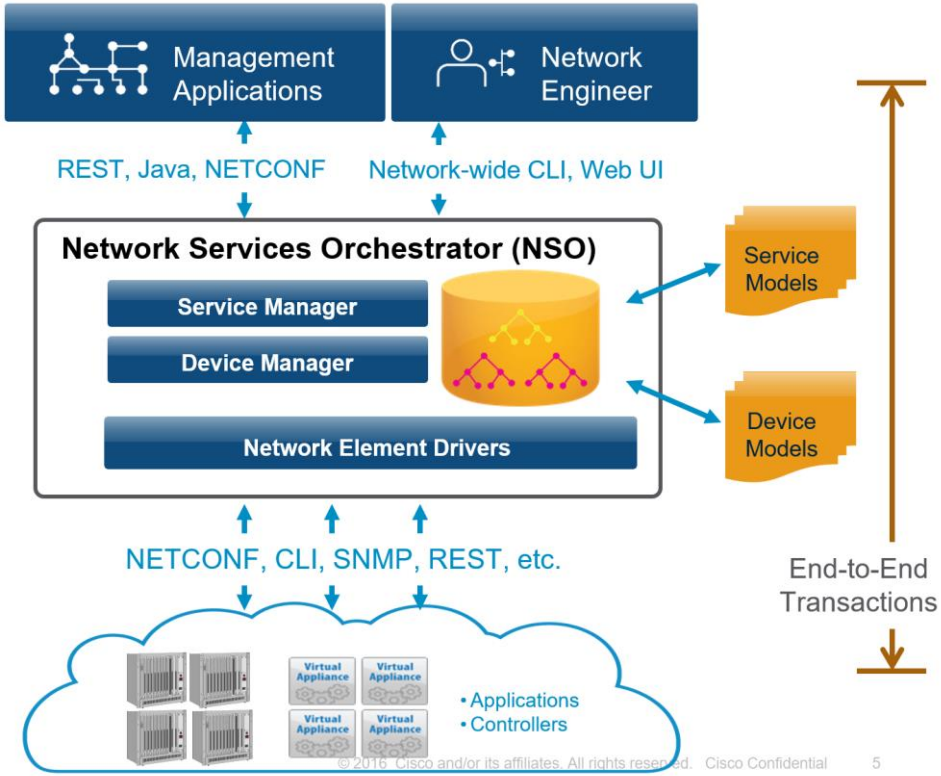
```
csr1000v#show ip access-lists
Standard IP access list OPTION1-ACL-in
    10 permit 192.168.1.0, wildcard bits 0.0.0.255
Standard IP access list OPTION2-ACL-in
    10 permit 192.168.2.0, wildcard bits 0.0.0.255
Standard IP access list OPTION3-ACL-in
    10 permit 192.168.3.0, wildcard bits 0.0.0.255
csr1000v#show runn int GigabitEthernet 3.3
Building configuration...

Current configuration : 203 bytes
!
interface GigabitEthernet3.3
 description OPTION3
 encapsulation dot1Q 3
 ip address 192.168.3.2 255.255.255.0
 ip access-group OPTION3-ACL-in in
 standby 1 ip 192.168.3.1
 standby 1 priority 120
end
```

Option 4: NSO + Python + IOS NED + CLI

Cisco Network Services Orchestrator (NSO)

- Enabled by Tail-F (acquired by Cisco in 2014)
- Multi-vendor service orchestrator for existing and future networks including:
  - Distributed (multi-device) service configuration management
  - Transaction integrity
  - Validation
  - Rollback
- YANG Model Driven Orchestration



NSO Service
Service Model

```
module: option4-nso-svc
   +--rw option4-nso-svc* [name]
      +--rw name                          string
...
      +--rw device                        -> /ncs:devices/device/name
      +--rw (interface)
      |  +--:(GigabitEthernet)
      |     +--rw GigabitEthernet
      |        +--rw name?   ->
/ncs:devices/device[ncs:name=current()/../../device]/config/ios:interface/GigabitEthernet/name
      +--rw ip-prefix?                    inet:ip-prefix
```

NSO Service
Service Template

```
<config-template xmlns="http://tail-f.com/ns/config/1.0">
  <devices xmlns="http://tail-f.com/ns/ncs">
    <device>
      <name>{/device}</name>
      <config>
        <ip xmlns="urn:ios">
          <access-list>
            <standard>
              <std-named-acl>
                <name>{/name}-ACL-in</name>
                <std-access-list-rule>
                  <rule>permit {$CIDR} {$WILDCARD}</rule>
                </std-access-list-rule>
              </std-named-acl>
            </standard>
          </access-list>
        </ip>
        <router xmlns="urn:ios">
          <ospf>
            <id>1</id>
            <network>
              <ip>{$CIDR}</ip>
              <mask>{$WILDCARD}</mask>
              <area>1</area>
            </network>
          </ospf>
        </router>
```

```
          <interface xmlns="urn:ios">
           <GigabitEthernet>
             <name>{$IF-NUM}.{$SUBIF}</name>
             <description>{/name}</description>
             <encapsulation>
               <dot1Q>
                 <vlan-id>{$SUBIF}</vlan-id>
               </dot1Q>
             </encapsulation>
             <ip>
               <access-group>
                 <direction>in</direction>
                 <access-list>{/name}-ACL-in</access-list>
               </access-group>
               <address>
                 <primary>
                   <address>{$IP2}</address>
                   <mask>{$MASK}</mask>
                 </primary>
               </address>
             </ip>
             <standby>
               <standby-list>
                 <group-number>1</group-number>
                 <ip>
                   <address>{$IP1}</address>
                 </ip>
                 <priority>120</priority>
               </standby-list>
             </standby>
           </GigabitEthernet>
         </interface>
       </config>
     </device>
   </devices>
</config-template>
```

NSO Service
Service Code

```
class Option4(Service):

    @Service.create
    def cb_create(self, tctx, root, service, proplist):
        self.log.info('Service create(service=', service._path, ')')

        vars = ncs.template.Variables()

        netaddr_prefix = netaddr.IPNetwork(service.ip_prefix)
        vars.add('CIDR',str(netaddr_prefix.network))
        vars.add('IP1',str(list(netaddr_prefix)[1]))
        vars.add('IP2',str(list(netaddr_prefix)[2]))
        vars.add('MASK',str(netaddr_prefix.netmask))
        vars.add('WILDCARD',str(netaddr_prefix.hostmask))

        if service.GigabitEthernet:
            vars.add('IF-NUM',service.GigabitEthernet.name)

vars.add('SUBIF',str(option4_find_next_subif(root,service.device,'GigabitEthernet',service.GigabitEt
hernet.name)))

        template = ncs.template.Template(service)
        template.apply('option4-nso-svc-template', vars)
```

Python Code
Running NSO Service

```
def create_service(arg_svc_name, arg_if_type, arg_if_num, arg_prefix, arg_dryrun=False):

    if nso_device is None:
        print("ERROR: Cannot map host into NSO device")
        return None

    restconf = False

    nso_device_sync_from(nso_device, restconf)

    if restconf is True:
        url = "data/option4-nso-svc={}{}".format(arg_svc_name, "?dryrun" if arg_dryrun is True else "")
    else:
        url = "running/option4-nso-svc/{}{}".format(arg_svc_name, "?dryrun=native" if arg_dryrun is True else "")

    dict = {"option4-nso-svc:option4-nso-svc": [{"name": arg_svc_name,
                                                 "device": nso_device,
                                                 "GigabitEthernet": {"name": arg_if_num},
                                                 "ip-prefix": arg_prefix
                                                 }]}

    json_payload = json.dumps(dict, sort_keys=False)

    response = nso_put(url, json_payload, restconf)

    if response.status_code == 201:
        if arg_dryrun is True:
            json_response = response.json()
            print(json_response['dryrun-result']['native']['device'][0]['data'])
        return ''
    else:
        print("ERROR: service not created, error ({})".format(response.text))
        return None
```

Option 4 Demo
Demo Service Instance

```
dvulovic@DVULOVIC-V2TQV:~/4o4na$ cat demosvc4.yml
---
parameters:
  service_name: OPTION4
  device: devnet_ios_xe
  interface_type: GigabitEthernet
  interface_number: 3
  ip_prefix: 192.168.4.0/24
```

Option 4 Demo
Run Script in “dry-run” Mode

```
dvulovic@DVULOVIC-V2TQV:~/4o4na$ ./create_service.py -s demosvc4.yml -o 4 -y

ip access-list standard OPTION4-ACL-in
 permit 192.168.4.0 0.0.0.255
!
interface GigabitEthernet3.4
 description OPTION4
 encapsulation dot1Q 4
 standby 1 ip 192.168.4.1
 standby 1 priority 120
 no switchport
 ip address 192.168.4.2 255.255.255.0
 ip access-group OPTION4-ACL-in in
 no shutdown
exit
router ospf 1
 network 192.168.4.0 0.0.0.255 area 1
exit
```

Option 4 Demo
Run Script

```
dvulovic@DVULOVIC-V2TQV:~/4o4na$ ./create_service.py -s demosvc4.yml -o 4
Service created.
```

Option 4 Demo
Check Results (DevNet IOS XE Sandbox)

```
csr1000v#show ip access-lists
Standard IP access list OPTION1-ACL-in
    10 permit 192.168.1.0, wildcard bits 0.0.0.255
Standard IP access list OPTION2-ACL-in
    10 permit 192.168.2.0, wildcard bits 0.0.0.255
Standard IP access list OPTION3-ACL-in
    10 permit 192.168.3.0, wildcard bits 0.0.0.255
Standard IP access list OPTION4-ACL-in
    10 permit 192.168.4.0, wildcard bits 0.0.0.255
csr1000v#show runn interface Gi 3.4
Building configuration...

Current configuration : 203 bytes
!
interface GigabitEthernet3.4
 description OPTION4
 encapsulation dot1Q 4
 ip address 192.168.4.2 255.255.255.0
 ip access-group OPTION4-ACL-in in
 standby 1 ip 192.168.4.1
 standby 1 priority 120
end
```

Comparison of Options

Comparison of Options
Framework Features

Issue	Generic Python	Python w/ YDK-Py	Ansible	Cisco NSO
Service/configuration DB	No	No	No	Yes (NSO CDB)
Formal definition of service parameters	No	No	No	Yes (YANG)
Configuration Templates	Optional (Jinja2)	No	Yes (Jinja2)	Yes (XML)
CLI Support	No	No	Yes (Network modules)	Yes (Network Element Drivers)
Automatic Buildout of Templates for Update	No	No	No	Yes (FASTMAP)
Free/Open Source	Yes	Yes	Yes	No

"Yes" = supported by the framework

"No" = not supported by the framework i.e. requires custom development

Comparison of Options
Subjective Recommendation

Issue	Generic Python	Python w/ YDK-Py	Ansible	Cisco NSO
Best to use for	Automation tasks of low complexity Full-blown product development		Automation tasks of low algorithmic complexity	All but very simple automation tasks
Not so best to use for	Automation tasks of medium/high complexity		Automation tasks of: medium/high algorithmic complexity	Very simple automation tasks


CISCO