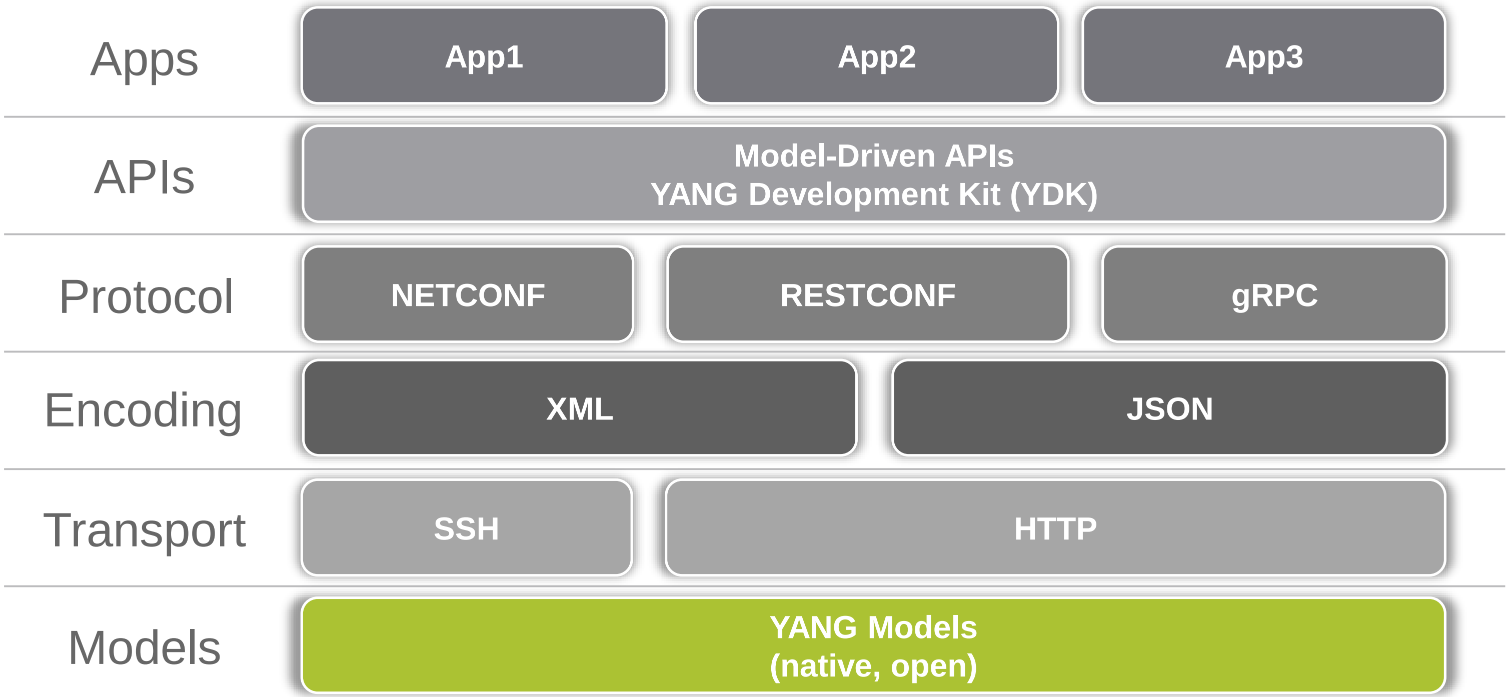
Model-Driven Programming with YDK

https://github.com/djordjevulovic/Collaborative_Intelligence_RS_2017

Djordje Vulovic
Consulting Systems Engineer, CCIE Emeritus
dvulovic@cisco.com

Model-Driven Programmability Stack



Apps: App1, App2, App3

APIs: Model-Driven APIs
YANG Development Kit (YDK)

Protocol: NETCONF, RESTCONF, gRPC

Encoding: XML, JSON

Transport: SSH, HTTP

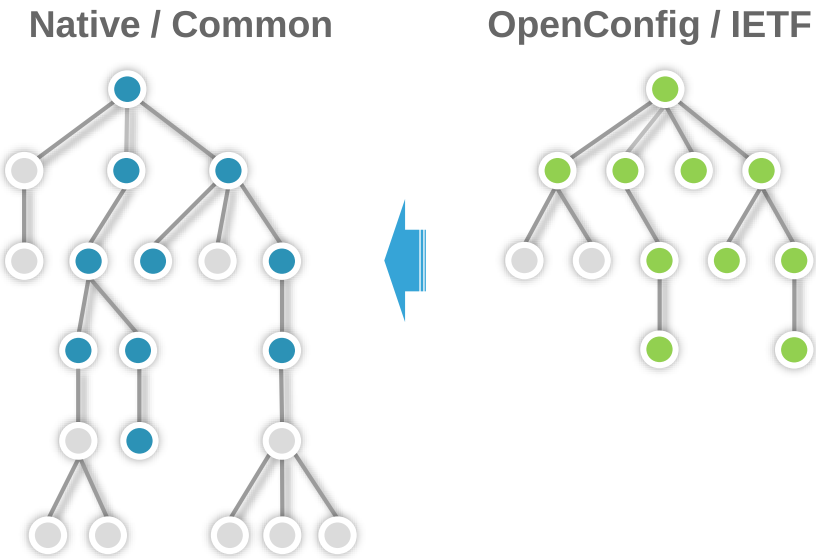
Models: YANG Models
(native, open)

Benefits of Model-Driven Programmability

- Model based, structured, computer friendly
- Multiple model types (native, common, OpenConfig, IETF, etc.)
- Models decoupled from transport, protocol end encoding
- Choice of transport, protocol and encoding
- Model-driven APIs for abstraction and simplification
- Wide standard support while leveraging open source

© 2016 Cisco and/or its affiliates. All rights reserved. Cisco Confidential

Data Models



Native / Common

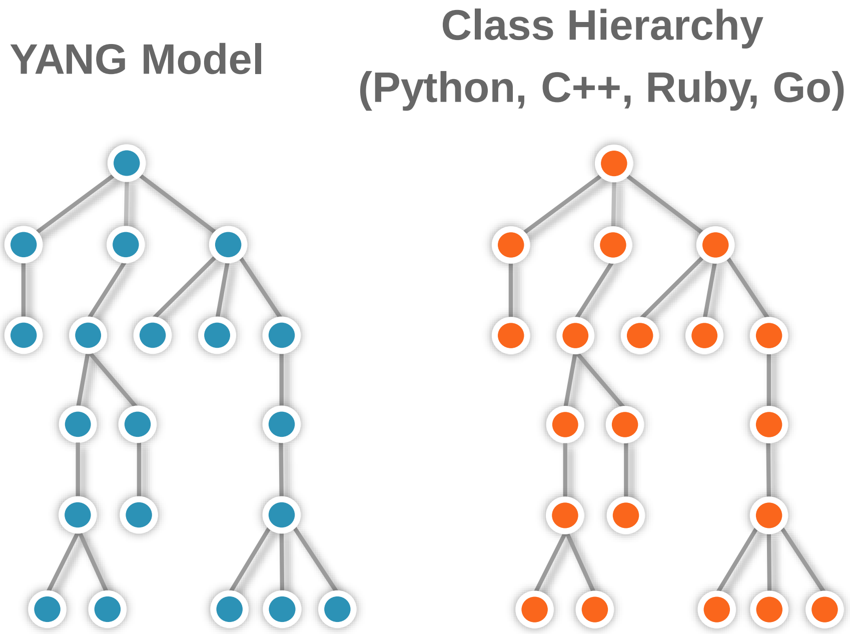
OpenConfig / IETF

- Model structures data (config and operational) as a tree
- Models files are self-documented and ship with devices
- Cisco IOS XR supports 180+ YANG models (config and operational) in release 6.0.1
- Native models provide most coverage
- OpenConfig support: BGP, Routing Policy, MPLS, Interfaces
- OpenConfig and IETF models are mapped to native models

© 2016 Cisco and/or its affiliates. All rights reserved. Cisco Confidential

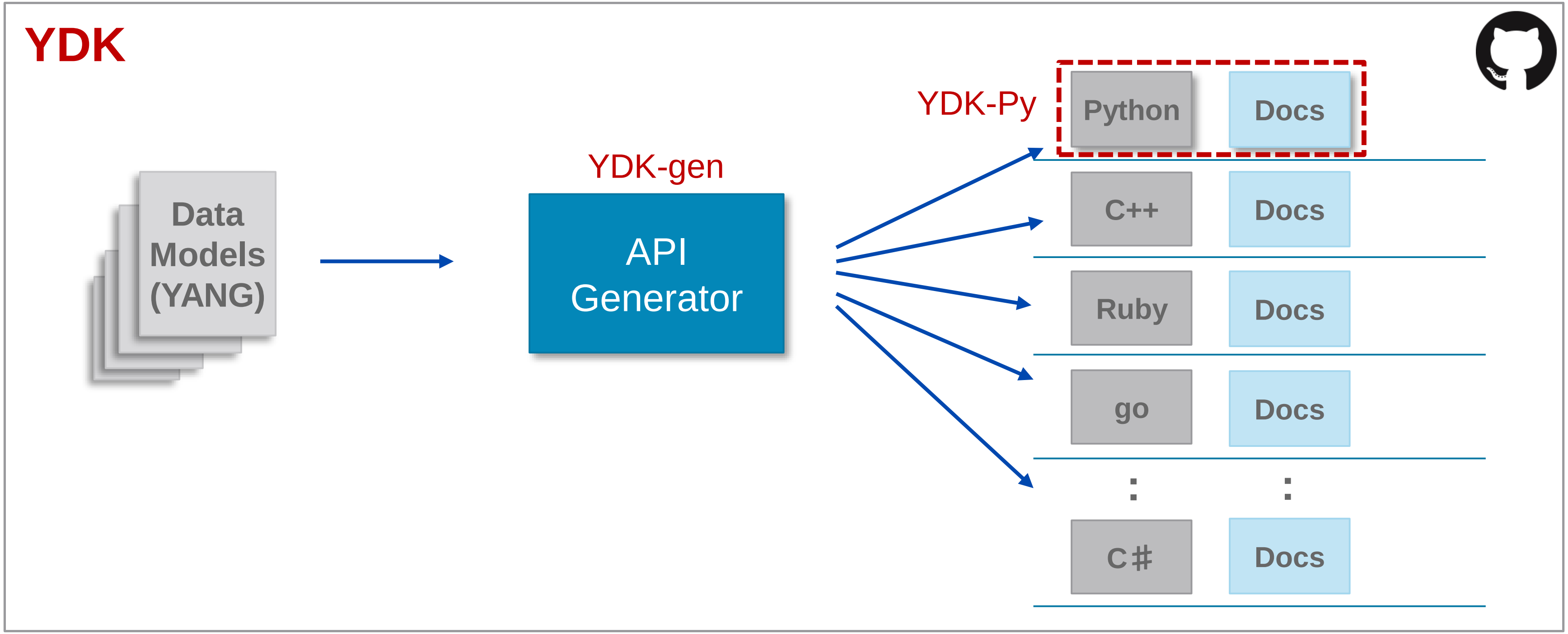
Model-Driven APIs

- 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.)



© 2016 Cisco and/or its affiliates. All rights reserved. Cisco Confidential

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



© 2016 Cisco and/or its affiliates. All rights reserved. Cisco Confidential

YDK Demo

© 2016 Cisco and/or its affiliates. All rights reserved. Cisco Confidential

YDK Demo App

- Available from the Github:
  - https://github.com/djordjevulovic/Collaborative_Intelligence_RS_2017
- If you are using Pycharm, refer to https://www.jetbrains.com/help/pycharm/2017.1/cloning-a-repository-from-github.html for instructions

© 2016 Cisco and/or its affiliates. All rights reserved. Cisco Confidential

Requirements

- dCloud YDK Sandbox v2 Lab
  - https://dcloud2-sjc.cisco.com/content/demo/2574?returnPathTitleKey=content-view
- Python
  - https://www.python.org/downloads/
- PyCharm (optional by highly recommended)
  - https://www.jetbrains.com/pycharm/download/
- YDK
  - “pip install ydk-models-cisco-ios-xr” (see next slide for Windows)

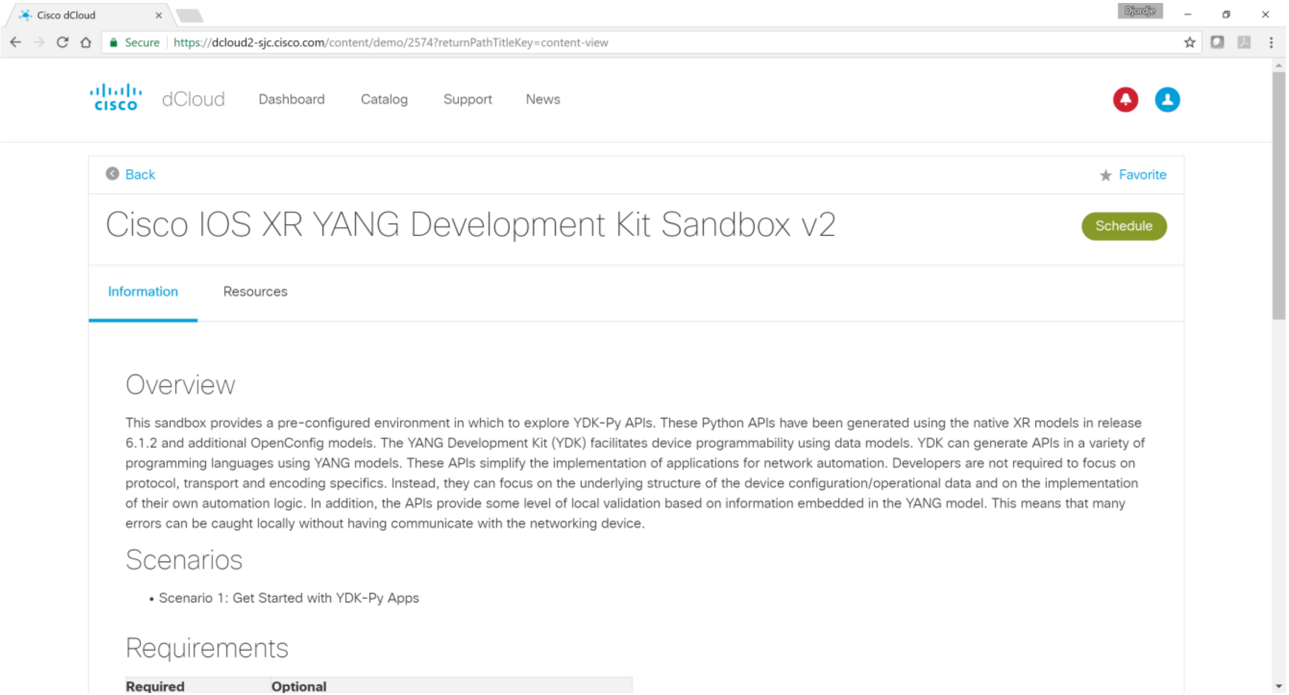
© 2016 Cisco and/or its affiliates. All rights reserved. Cisco Confidential

Installing YDK on Windows

- YDK-PY by default requires exact version of lxml library (3.4.4)
- On Windows, building this package will likely fail
- Workaround:
  - Download YDK Source (e.g. https://github.com/CiscoDevNet/ydk-py/archive/master.zip)
  - In setup.py files, change from 'lxml==3.4.4‘ to 'lxml>=3.4.4‘
  - Install YDK from source (https://github.com/CiscoDevNet/ydk-py#installing-from-source)

© 2016 Cisco and/or its affiliates. All rights reserved. Cisco Confidential

dCloud Lab – YDK Sandbox v2 Lab



Cisco IOS XR YANG Development Kit Sandbox v2

Overview

This sandbox provides a pre-configured environment in which to explore YDK-Py APIs. These Python APIs have been generated using the native XR models in release 6.1.2 and additional OpenConfig models. The YANG Development Kit (YDK) facilitates device programmability using data models. YDK can generate APIs in a variety of programming languages using YANG models. These APIs simplify the implementation of applications for network automation. Developers are not required to focus on protocol, transport and encoding specifics. Instead, they can focus on the underlying structure of the device configuration/operational data and on the implementation of their own automation logic. In addition, the APIs provide some level of local validation based on information embedded in the YANG model. This means that many errors can be caught locally without having communicate with the networking device.

Scenarios

- Scenario 1: Get Started with YDK-Py Apps

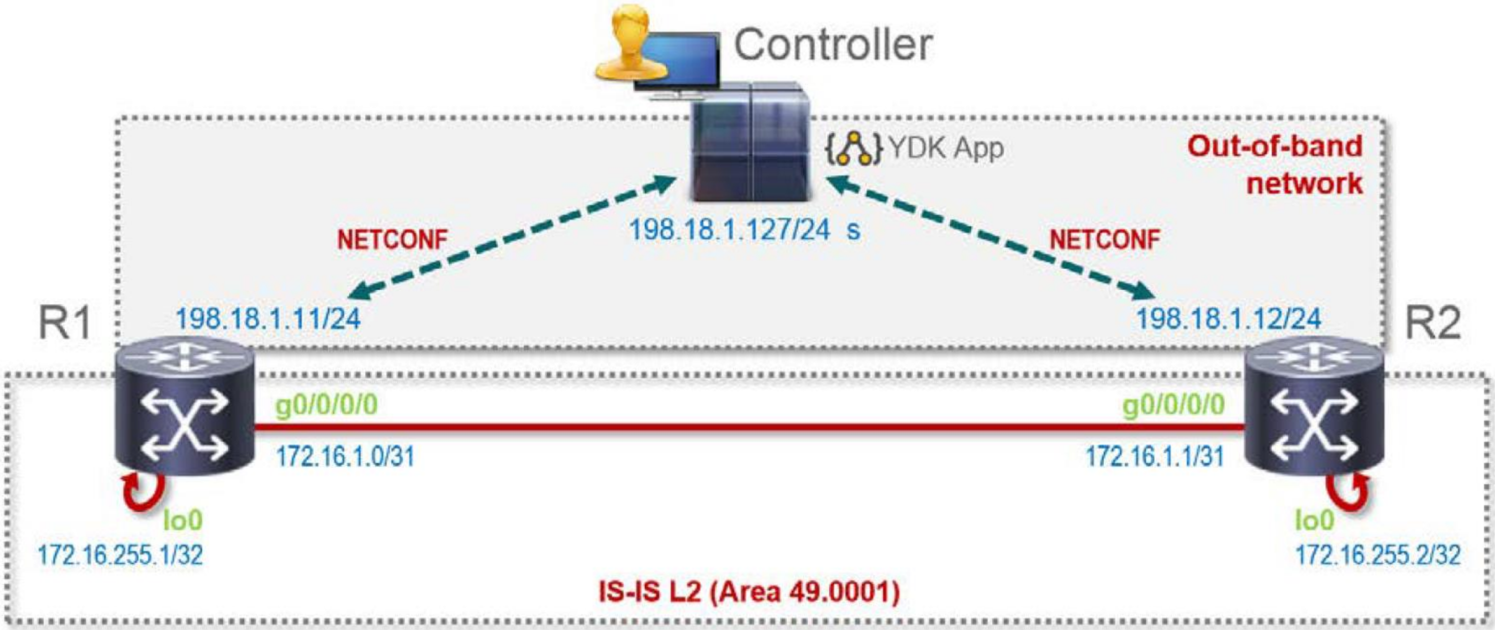
Requirements

Required	Optional

https://dcloud2-sjc.cisco.com/content/demo/2574?returnPathTitleKey=content-view

© 2016 Cisco and/or its affiliates. All rights reserved. Cisco Confidential

YDK SandBox Lab - Topology



Application Workflow

- Step #1 – Create Loopback Interface
  - Lo 1111 with IP 1.1.1.1/24 on R1
  - Lo 2222 with IP 2.2.2.2/24 on R2
- Step #2 – Create BGP Process
  - AS 65000
- Step #3 – Create BGP Neighbor
  - Peer address is respective Loopback 0 IP address
- Step #4 – Advertise Network
  - 1.1.1.0/24 (R1) and 2.2.2.0/24 (R2)
- Step #5 – Add IPv4/Unicast SAFI to BGP Neighbor

© 2016 Cisco and/or its affiliates. All rights reserved. Cisco Confidential

YANG Models Used for Configuration

Step	R1	R2
Step #1	IOS XR Native	OpenConfig
Step #2	IOS XR Native	OpenConfig
Step #3	IOS XR Native	OpenConfig
Step #4	IOS XR Native	IOS XR Native
Step #5	IOS XR Native	OpenConfig

© 2016 Cisco and/or its affiliates. All rights reserved. Cisco Confidential

Step #1 – Create Loopback

Target Configuration in IOS XR CLI

```
interface Loopback1111
 ipv4 address 1.1.1.1 255.255.255.0
```

© 2016 Cisco and/or its affiliates. All rights reserved. Cisco Confidential

Step #1 – Create Loopback

Cisco-IOS-XR-ifmgr-cfg YANG Model (part of)

```
 container interface-configurations {
    description "interface configurations";

    list interface-configuration {
      key "active interface-name";
      description "The configuration for an interface";

      leaf interface-name {
        type xr:Interface-name;
        description "The name of the interface";
      }

      leaf interface-virtual {
        type empty;
        description
          "The mode in which an interface is running. The
          existence of this object causes the creation of
          the software virtual/subinterface.";
      }

      leaf active {
        type Interface-active;
        description
          "Whether the interface is active or
          preconfigured";
      }
      ….
    }
}
```

Step #1 – Create Loopback

Target Configuration in IOS XR Native Model

```
<interface-configurations xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ifmgr-cfg">
  <interface-configuration>
   <active>act</active>
   <interface-name>Loopback1111</interface-name>
   <interface-virtual/>
   <ipv4-network xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ipv4-io-cfg">
    <addresses>
     <primary>
      <address>1.1.1.1</address>
      <netmask>255.255.255.0</netmask>
     </primary>
    </addresses>
   </ipv4-network>
  </interface-configuration>
</interface-configurations>
```

© 2016 Cisco and/or its affiliates. All rights reserved.   Cisco Confidential

Step #1 – Create Loopback

Python class method for IOS XR Native Model

```
class dvulovic_Native_IOSXR_Model_YDK(dvulovic_Generic_IOSXR_Model):

    def create_loopback(self, arg_loopbacknum, arg_ip, arg_mask):
        interface_configurations = Cisco_IOS_XR_ifmgr_cfg.InterfaceConfigurations()

        interface_configuration = interface_configurations.InterfaceConfiguration()

        interface_configuration.active = "act"
        interface_configuration.interface_name = "Loopback" + arg_loopbacknum
        interface_configuration.interface_virtual = ydk.types.Empty()

        primary_address = interface_configuration.ipv4_network.addresses.Primary()
        primary_address.address = arg_ip
        primary_address.netmask = arg_mask

        interface_configuration.ipv4_network.addresses.primary = primary_address

        interface_configurations.interface_configuration.append(interface_configuration)

        self.xr.ydk_crud.create(self.xr.ydk_provider, interface_configurations)
```

Step #1 – Create Loopback

openconfig-interfaces.yang Model (part of)

```
 container interfaces {
      description
        "Top level container for interfaces, including configuration
        and state data.";


      list interface {
        key "name";

        description
          "The list of named interfaces on the device.";

        leaf name {
          type leafref {
            path "../config/name";
          }
        }

        container config {
          description
            "Configurable items at the global, physical interface
            level";

          uses interface-phys-config;
        }
}
```

© 2016 Cisco and/or its affiliates. All rights reserved. Cisco Confidential

Step #1 – Create Loopback

Target Configuration in OpenConfig Model

```
<interfaces xmlns="http://openconfig.net/yang/interfaces">
  <interface>
   <name>Loopback1111</name>
   <config>
    <name>Loopback1111</name>
    <type xmlns:idx="urn:ietf:params:xml:ns:yang:iana-if-type">idx:softwareLoopback</type>
    <enabled>true</enabled>
   </config>
   <subinterfaces>
    <subinterface>
     <index>0</index>
     <ipv4 xmlns="http://openconfig.net/yang/interfaces/ip">
      <address>
       <ip>1.1.1.1</ip>
       <config>
        <ip>1.1.1.1</ip>
        <prefix-length>24</prefix-length>
       </config>
      </address>
     </ipv4>
    </subinterface>
   </subinterfaces>
  </interface>
</interfaces>
```

Step #1 – Create Loopback

Python class method for OpenConfig Model

```
class dvulovic_OpenConfig_IOSXR_Model_YDK(dvulovic_Generic_IOSXR_Model):

    def create_loopback(self, arg_loopbacknum, arg_ip, arg_prefixlen):
            oc_interface = OpenConfig_Interfaces.Interface()

            oc_interface.name = "Loopback" + arg_loopbacknum
            oc_interface.config.name = "Loopback" + arg_loopbacknum
            oc_interface.config.type = SoftwareloopbackIdentity()
            oc_interface.config.enabled = True

            oc_subinterface = oc_interface.subinterfaces.Subinterface()
            oc_subinterface.index = 0

            oc_subinterface_ipv4 = oc_subinterface.Ipv4()

            oc_subinterface_ipv4_address = oc_subinterface_ipv4.Address()
            oc_subinterface_ipv4_address.ip = arg_ip
            oc_subinterface_ipv4_address.config.ip = arg_ip
            oc_subinterface_ipv4_address.config.prefix_length = arg_prefixlen
            oc_subinterface_ipv4.address.append(oc_subinterface_ipv4_address)

            oc_subinterface.ipv4 = oc_subinterface_ipv4

            oc_interface.subinterfaces.subinterface.append(oc_subinterface)

            self.xr.ydk_crud.create(self.xr.ydk_provider, oc_interface)
```

© 2016 Cisco and/or its affiliates. All rights reserved. Cisco Confidential

Step #1 – Create Loopback

Python code to run step #1

```
# step 1 - create loopback interface
xr_native_model_r1.create_loopback("1111", "1.1.1.1", "255.255.255.0")
xr_oc_model_r2.create_loopback("2222","2.2.2.2", 24)
```

© 2016 Cisco and/or its affiliates. All rights reserved.

Step #1 – Create Loopback

Generated IOS XR CLI (R1)

```
RP/0/RP0/CPU0:r1#show configuration commit changes last 1
Thu Jan 26 13:21:12.899 UTC
Building configuration...
!! IOS XR Configuration version = 6.1.2
interface Loopback1111
 ipv4 address 1.1.1.1 255.255.255.0
!
end
```

Step #1 – Create Loopback

Generated IOS XR CLI (R2)

```
RP/0/RP0/CPU0:r2#show configuration commit changes last 1
Thu Jan 26 11:23:00.071 UTC
Building configuration...
!! IOS XR Configuration version = 6.1.2
interface Loopback2222
 ipv4 address 2.2.2.2 255.255.255.0
!
end
```

© 2016 Cisco and/or its affiliates. All rights reserved. Cisco Confidential

Step #2 – Create BGP Process

Python code to run step #2

```
# step 2 - create BGP process
xr_native_model_r1.create_bgp_process(65000)
xr_oc_model_r2.create_bgp_procces(65000)
```

Step #2 – Create BGP Process

Generated IOS XR CLI (R1)

```
RP/0/RP0/CPU0:r1#show configuration commit changes last 1
Thu Jan 26 13:24:44.804 UTC
Building configuration...
!! IOS XR Configuration version = 6.1.2
router bgp 65000
!
end

RP/0/RP0/CPU0:r1#
```

© 2016 Cisco and/or its affiliates. All rights reserved. Cisco Confidential

Step #2 – Create BGP Process

Generated IOS XR CLI (R2)

```
RP/0/RP0/CPU0:r2#show configuration commit changes last 1
Thu Jan 26 11:25:23.205 UTC
Building configuration...
!! IOS XR Configuration version = 6.1.2
router bgp 65000
!
end

RP/0/RP0/CPU0:r2#
```

© 2016 Cisco and/or its affiliates. All rights reserved. Cisco Confidential

Step #3 – Create BGP Neighbor

Target Configuration in IOS XR CLI

```
 router bgp 65000
 !
 neighbor 172.16.255.2
  remote-as 65000
  update-source Loopback0
  !
 !
!
```

© 2016 Cisco and/or its affiliates. All rights reserved. Cisco Confidential

Step #3 – Create BGP Neighbor

Target Configuration in IOS XR Native Model

```
<bgp xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ipv4-bgp-cfg">
  <instance>
   <instance-name>default</instance-name>
   <instance-as>
    <as>0</as>
    <four-byte-as>
     <as>65000</as>
     <bgp-running/>
     <default-vrf>
      <bgp-entity>
       <neighbors>
        <neighbor>
         <neighbor-address>172.16.255.2</neighbor-address>
         <remote-as>
          <as-xx>0</as-xx>
          <as-yy>65000</as-yy>
         </remote-as>
         <update-source-interface>Loopback0</update-source-interface>
        </neighbor>
       </neighbors>
      </bgp-entity>
     </default-vrf>
    </four-byte-as>
   </instance-as>
  </instance>
 </bgp>
```

Step #3 – Create BGP Neighbor

Target Configuration in OpenConfig Model

```
<bgp xmlns="http://openconfig.net/yang/bgp">
  <global>
   <config>
    <as>65000</as>
   </config>
  </global>
  <neighbors>
   <neighbor>
    <neighbor-address>172.16.255.2</neighbor-address>
    <config>
     <neighbor-address>172.16.255.2</neighbor-address>
     <peer-as>65000</peer-as>
    </config>
    <transport>
     <config>
      <local-address>Loopback0</local-address>
     </config>
    </transport>
   </neighbor>
  </neighbors>
 </bgp>
```

Step #3 – Create BGP Neighbor

Python code to run step #3

```
# step 3 - create bgp neighbor
xr_native_model_r1.add_bgp_neighbor(65000,"172.16.255.2",65000, "Loopback0")
xr_oc_model_r2.add_bgp_neighbor(65000,"172.16.255.1",65000,"Loopback0")
```

© 2016 Cisco and/or its affiliates. All rights reserved.

Step #3 – Create BGP Neighbor

Generated IOS XR CLI (R1)

```
RP/0/RP0/CPU0:r1#show configuration commit changes last 1
Thu Jan 26 13:26:26.503 UTC
Building configuration...
!! IOS XR Configuration version = 6.1.2
router bgp 65000
 neighbor 172.16.255.2
  remote-as 65000
  update-source Loopback0
 !
!
end
```

© 2016 Cisco and/or its affiliates. All rights reserved. Cisco Confidential

Step #3 – Create BGP Neighbor

Generated IOS XR CLI (R2)

```
RP/0/RP0/CPU0:r2#show configuration commit changes last 1
Thu Jan 26 11:26:22.691 UTC
Building configuration...
!! IOS XR Configuration version = 6.1.2
router bgp 65000
 neighbor 172.16.255.1
  remote-as 65000
  update-source Loopback0
 !
!
end
```

© 2016 Cisco and/or its affiliates. All rights reserved.   Cisco Confidential

Step #4 – Advertise Network

Python code to run step #4

```
# step 4 - advertise network
xr_native_model_r1.add_bgp_ipv4_unicast_network(65000,"1.1.1.0",24)
xr_native_model_r2.add_bgp_ipv4_unicast_network(65000,"2.2.2.0",24)
```

© 2016 Cisco and/or its affiliates. All rights reserved. Cisco Confidential

Step #4 – Advertise Network

Generated IOS XR CLI (R1)

```
RP/0/RP0/CPU0:r1#show configuration commit changes last 1
Thu Jan 26 13:30:43.825 UTC
Building configuration...
!! IOS XR Configuration version = 6.1.2
router bgp 65000
 address-family ipv4 unicast
  network 1.1.1.0/24
 !
!
end
```

Step #4 – Advertise Network

Generated IOS XR CLI (R2)

```
RP/0/RP0/CPU0:r2#show configuration commit changes last 1
Thu Jan 26 11:27:30.757 UTC
Building configuration...
!! IOS XR Configuration version = 6.1.2
router bgp 65000
 address-family ipv4 unicast
  network 2.2.2.0/24
 !
!
end
```

© 2016 Cisco and/or its affiliates. All rights reserved.

Step #5 – Add IPv4/Unicast SAFI

Python code to run step #5

```
# step 5 - add IPv4 Unicast SAFI to BGP neighbor
xr_native_model_r1.add_ipv4_unicast_SAFI_to_bgp_neighbor(65000, "172.16.255.2")
xr_oc_model_r2.add_ipv4_unicast_SAFI_to_bgp_neighbor(65000,"172.16.255.1")
```

Step #5 – Add IPv4/Unicast SAFI

Generated IOS XR CLI (R1)

```
RP/0/RP0/CPU0:r1#show configuration commit changes last 1
Thu Jan 26 13:31:26.675 UTC
Building configuration...
!! IOS XR Configuration version = 6.1.2
router bgp 65000
 neighbor 172.16.255.2
  address-family ipv4 unicast
  !
 !
!
end
```

© 2016 Cisco and/or its affiliates. All rights reserved.   Cisco Confidential

Step #5 – Add IPv4/Unicast SAFI

Generated IOS XR CLI (R2)

```
RP/0/RP0/CPU0:r2#show configuration commit changes last 1
Thu Jan 26 11:29:00.024 UTC
Building configuration...
!! IOS XR Configuration version = 6.1.2
router bgp 65000
 neighbor 172.16.255.1
  address-family ipv4 unicast
  !
 !
!
end
```

The Result

BGP Table (R1)

```
RP/0/RP0/CPU0:r1#show bgp
Thu Jan 26 13:45:55.397 UTC
BGP router identifier 172.16.255.1, local AS number 65000
BGP generic scan interval 60 secs
Non-stop routing is enabled
BGP table state: Active
Table ID: 0xe0000000   RD version: 3
BGP main routing table version 3
BGP NSR Initial initsync version 1 (Reached)
BGP NSR/ISSU Sync-Group versions 0/0
BGP scan interval 60 secs

Status codes: s suppressed, d damped, h history, * valid, > best
              i - internal, r RIB-failure, S stale, N Nexthop-discard
Origin codes: i - IGP, e - EGP, ? - incomplete
   Network            Next Hop            Metric LocPrf Weight Path
*> 1.1.1.0/24         0.0.0.0                  0         32768 i
*>i2.2.2.0/24         172.16.255.2             0    100      0 i

Processed 2 prefixes, 2 paths
```

The Result

BGP Table (R2)

```
RP/0/RP0/CPU0:r2#show bgp
Thu Jan 26 11:30:58.017 UTC
BGP router identifier 172.16.255.2, local AS number 65000
BGP generic scan interval 60 secs
Non-stop routing is enabled
BGP table state: Active
Table ID: 0xe0000000   RD version: 3
BGP main routing table version 3
BGP NSR Initial initsync version 1 (Reached)
BGP NSR/ISSU Sync-Group versions 0/0
BGP scan interval 60 secs

Status codes: s suppressed, d damped, h history, * valid, > best
              i - internal, r RIB-failure, S stale, N Nexthop-discard
Origin codes: i - IGP, e - EGP, ? - incomplete
   Network            Next Hop            Metric LocPrf Weight Path
*>i1.1.1.0/24         172.16.255.1             0    100      0 i
*> 2.2.2.0/24         0.0.0.0                  0         32768 i

Processed 2 prefixes, 2 paths
```

The Result

Routing Tables (R1, R2)

```
RP/0/RP0/CPU0:r1#sh ip ro 2.2.2.0
Thu Jan 26 13:48:18.808 UTC

Routing entry for 2.2.2.0/24
  Known via "bgp 65000", distance 200, metric 0, type internal
  Installed Jan 26 13:44:44.525 for 00:03:35
  Routing Descriptor Blocks
    172.16.255.2, from 172.16.255.2
      Route metric is 0
  No advertising protos.
```

```
RP/0/RP0/CPU0:r2#sh ip ro 1.1.1.0
Thu Jan 26 11:33:10.905 UTC

Routing entry for 1.1.1.0/24
  Known via "bgp 65000", distance 200, metric 0, type internal
  Installed Jan 26 11:30:03.361 for 00:03:09
  Routing Descriptor Blocks
    172.16.255.1, from 172.16.255.1
      Route metric is 0
  No advertising protos.
```

© 2016 Cisco and/or its affiliates. All rights reserved. Cisco Confidential

The Result

Ping (R1, R2)

```
RP/0/RP0/CPU0:r1#ping 2.2.2.2 so 1.1.1.1
Thu Jan 26 13:48:49.520 UTC
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 2.2.2.2, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 3/3/5 ms
RP/0/RP0/CPU0:r1#
```

```
RP/0/RP0/CPU0:r2#ping 1.1.1.1 so 2.2.2.2
Thu Jan 26 11:34:48.511 UTC
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 1.1.1.1, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 4/5/9 ms
```

© 2016 Cisco and/or its affiliates. All rights reserved. Cisco Confidential

cisco

Step #1 – Create Loopback

Target Configuration in IOS XR Native Model

```
<interface-configurations xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ifmgr-cfg">
  <interface-configuration>
   <active>act</active>
   <interface-name>Loopback1111</interface-name>
   <interface-virtual/>
   <ipv4-network xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ipv4-io-cfg">
    <addresses>
     <primary>
      <address>1.1.1.1</address>
      <netmask>255.255.255.0</netmask>
     </primary>
    </addresses>
   </ipv4-network>
  </interface-configuration>
</interface-configurations>
```

© 2016 Cisco and/or its affiliates. All rights reserved.

Step #1 – Create Loopback

Target Configuration in OpenConfig Model

```
<interfaces xmlns="http://openconfig.net/yang/interfaces">
  <interface>
   <name>Loopback1111</name>
   <config>
    <name>Loopback1111</name>
    <type xmlns:idx="urn:ietf:params:xml:ns:yang:iana-if-type">idx:softwareLoopback</type>
    <enabled>true</enabled>
   </config>
   <subinterfaces>
    <subinterface>
     <index>0</index>
     <ipv4 xmlns="http://openconfig.net/yang/interfaces/ip">
      <address>
       <ip>1.1.1.1</ip>
       <config>
        <ip>1.1.1.1</ip>
        <prefix-length>24</prefix-length>
       </config>
      </address>
     </ipv4>
    </subinterface>
   </subinterfaces>
  </interface>
</interfaces>
```

© 2016 Cisco and/or its affiliates. All rights reserved.   Cisco Confidential

Step #1 – Create Loopback

Target Configuration in OpenConfig Model

```
 container interfaces {
      description
        "Top level container for interfaces, including configuration
        and state data.";


      list interface {
        key "name";

        description
          "The list of named interfaces on the device.";

        leaf name {
          type leafref {
            path "../config/name";
          }
        }

        container config {
          description
            "Configurable items at the global, physical interface
            level";

          uses interface-phys-config;
        }
}
```

© 2016 Cisco and/or its affiliates. All rights reserved. Cisco Confidential