
cisco
You make possible

Network Automation Journey with Cisco Network Services Orchestrator (NSO)

Sunpreetsingh Arora CSE @sunpreetsinghar
Hector Oses CSE

TECNMS-4175

cisco Live!
Barcelona | January 27-31, 2020

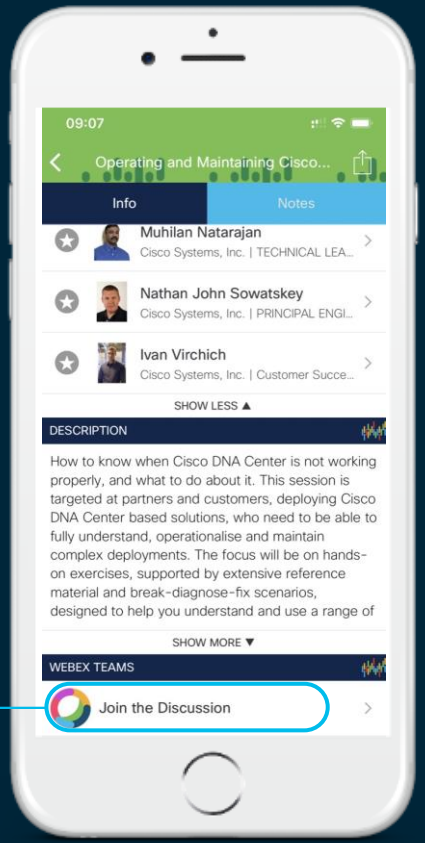
Cisco Webex Teams

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



The bridge between Network Automation and Business

Agenda

- Introduction
- Network Automation challenges
- NSO Architecture
  - Device Management + Lab
  - YANG
  - Service Manager
  - Package Manager
- Service Design + Lab

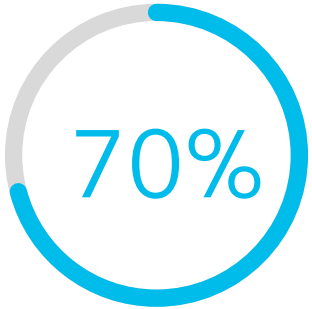
Sunpreetsingh Arora

Hector Oses

Network Automation Challenges

Business Challenges

Network Changes
Performed Manually



Policy Violations
Due to Human Error

OpEx Spent on Network
Changes and Troubleshooting

$60B Spent on Network Operations Labor and Tools

Source: McKinsey study conducted for Cisco in 2016

The automation drive

Complexity

- Increasing number of complex technologies
- Increasing scale
- Skills shortage

Quality

- Faster time to market
- Differentiation demand
- Security focus

Economics

- Reduce Opex
- Reduce human errors
- Account Role of technology investment with business outcomes

The Benefits of Automation

EFFICIENCY

+

AGILITY

+

DIGITAL TRANSFORMATION

IT COST REDUCTION
77%

ABILITY TO MEET SLAS
72%

TIME TO PROVISION IT SERVICES
87%

STRATEGIC ALLOCATION OF IT BUDGET
200%

REVENUE GROWTH
11%

Source: Source: IDC InfoBrief, sponsored by Cisco, Cloud Going Mainstream. All Are Trying, Some Are Benefiting; Few Are Maximizing Value. September 2016

Tie Islands of Automation Together
End-to-End Automation Delivers Business Value

Multiple Operating Systems and platforms (>10)

Different Types of Automation Tools (>50)

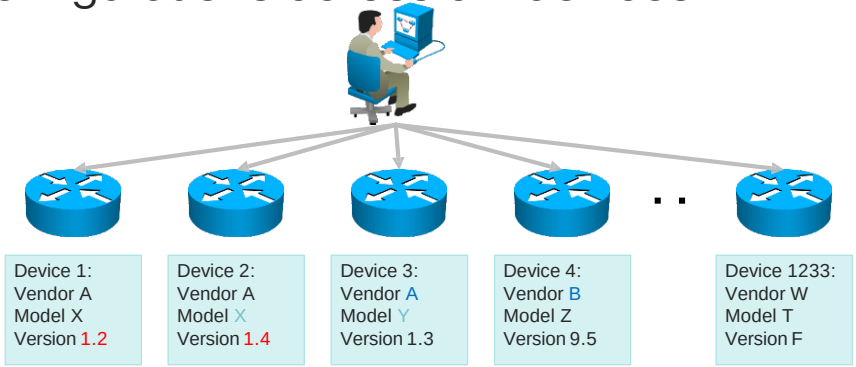
Scripts (>10,000)

Manual Activity (>1,000)

Orchestration software, cross-team collaboration

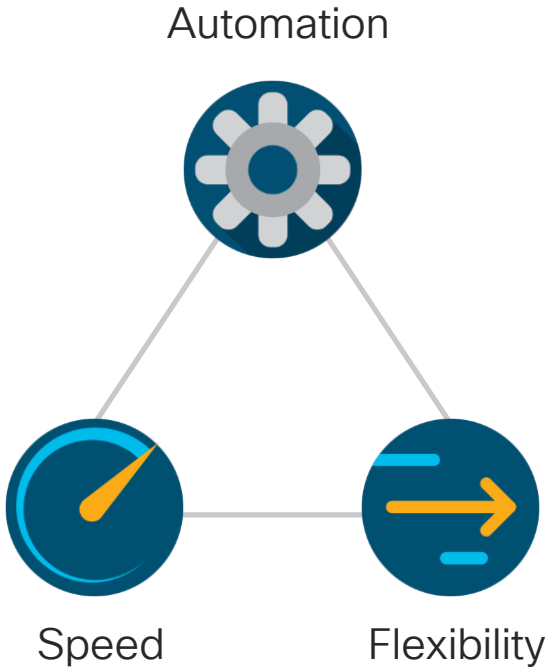
Network Management Challenges

- We are configuring **different** devices. - multi-vendor environment.
- There is no real **service** management
- There is no abstract **models** beyond device level
- Understand the capabilities and limitations of each device and device group
- Ensure consistency and reliability of configurations across all devices
- Backup and restore configurations.

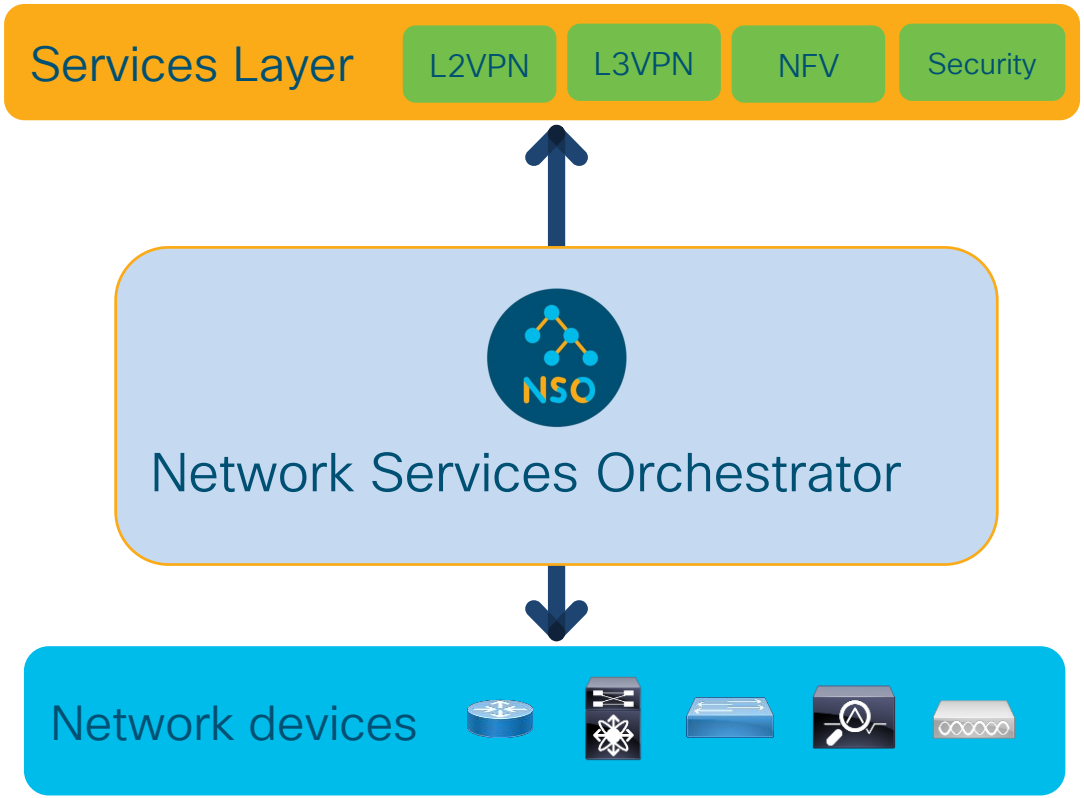


Simplifying complexity and improving quality

- Full Service Lifecycle Management
- Model Driven
- Real Time Services
- Multi Vendor, Multi Service Type Support
- Rich Set of Northbound API's
- Network-wide Unified (CLI)



Orchestration Architecture



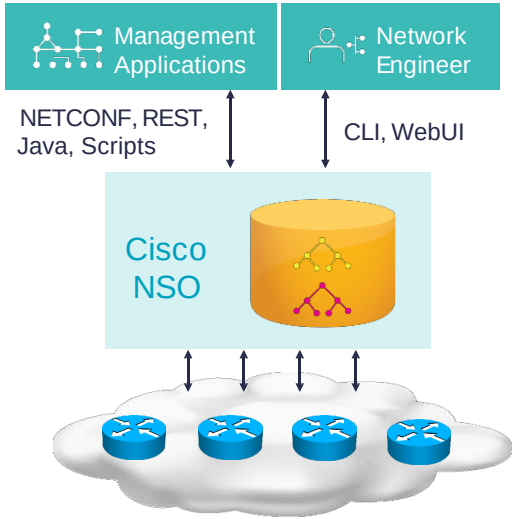
Quick History

- Founded in Sweden in 2005
- Acquired by Cisco in 2014
- Developed Conf-D and NCS
- NCS evolved into NSO!

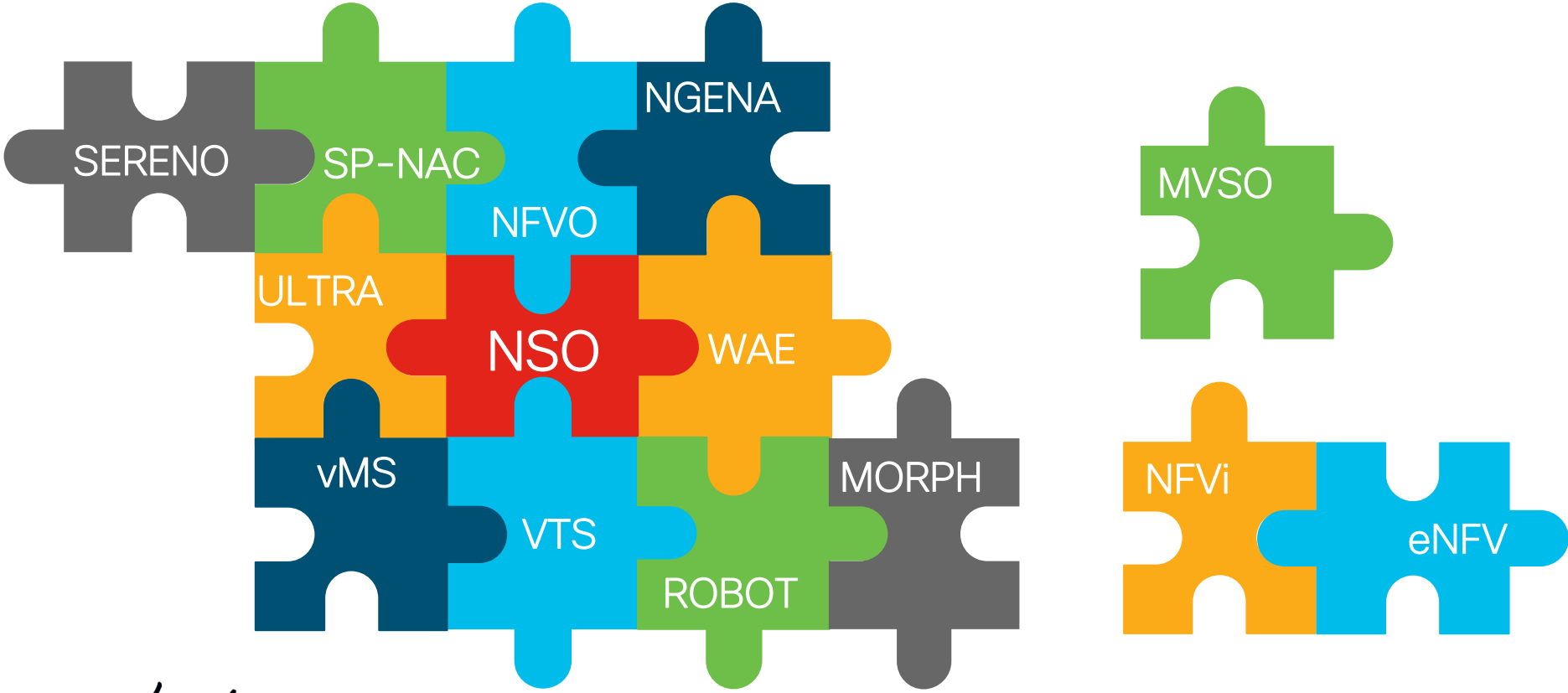
Addressing Network Management Challenges

Cisco NSO Solution

- Multi-vendor service orchestration platform
- Multi-vendor service-layer SDN controller
- Supports traditional L2-L7 networking, virtual devices and OpenFlow
- Provides a single API and single UI to entire managed environment
- Keeps accurate copy of network configuration state

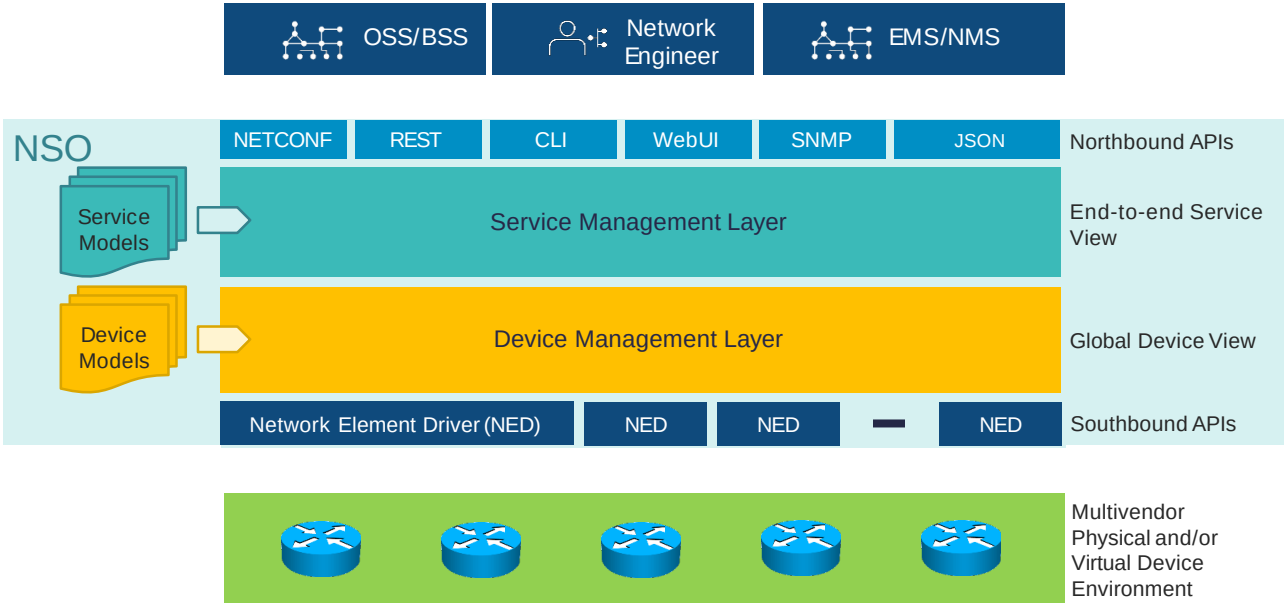


Solutions ecosystem

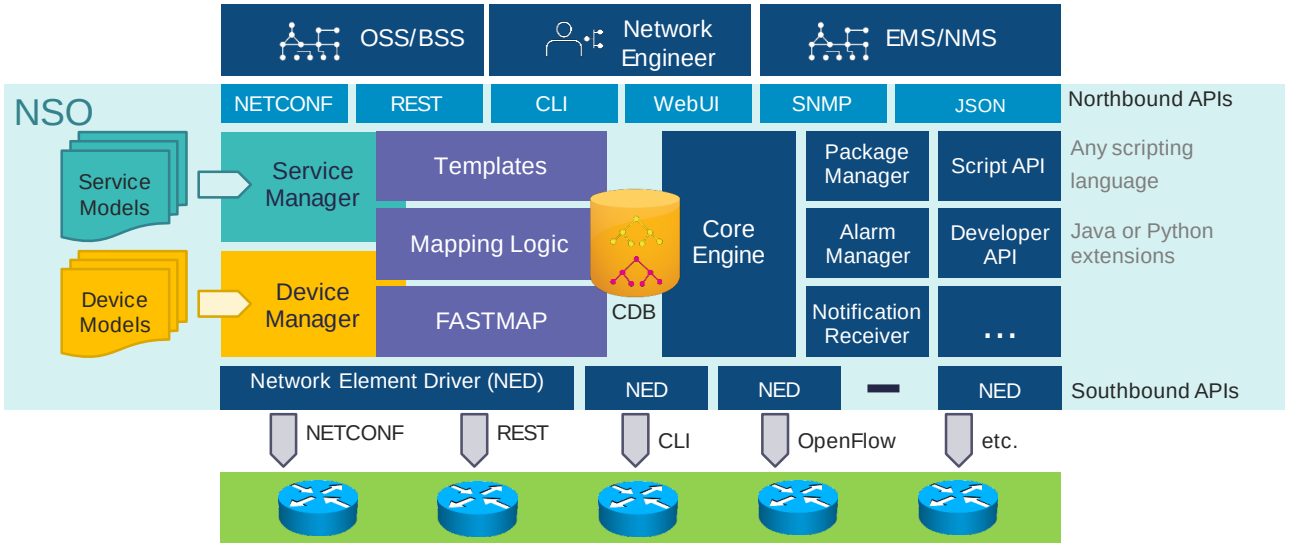


NSO Architecture

NSO Network Abstraction

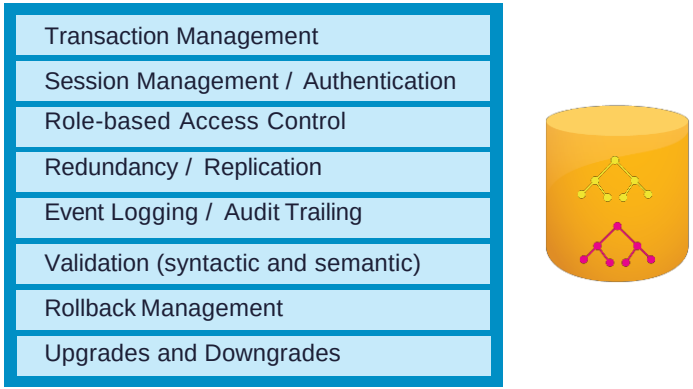


NSO Architectural Components



NSO components

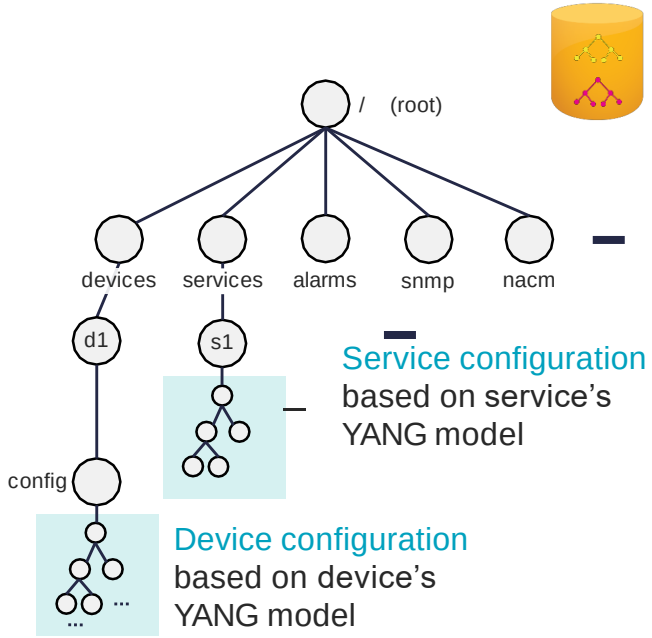
Cisco NSO Core Engine



- Performs core functionality
- Glue for all NSO components
- Reads initial configuration from *ncs.conf*
- All other configuration and operational data is handled in the CDB main configuration database (CDB)

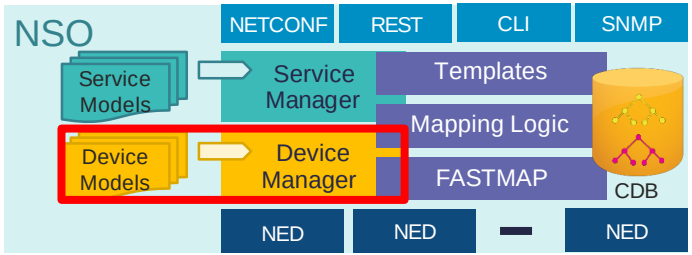
CDB

- Stores configuration data for devices and system
- ACID compliant
- Fast, lightweight, fault-tolerant
- Compact binary XML-analogue format
- In-memory DB with file system journaling for persistence
- 1:N data replication support
- In-service data model update
- Automatic schema and data update
- Automatic loading of initialization data



Device Manager

- Handles device configurations
  - Device configuration framework defined using YANG data models from NEDs
- Service models map to device configurations:
  - Configuration templates (XML)
  - Java or Python code
- FASTMAP algorithm controls and optimizes southbound configuration actions:
  - Create
  - Modify
  - Delete



NSO CLI

- In Operational mode, the CLI displays operational data stored in CDB
- In Configuration mode, the CLI displays network configuration data stored in CDB

Cisco Style

```
admin@nso# operational mode CLI

admin@nso(config)# configuration mode
```

Juniper Style

```
admin@nso> operational mode CLI

admin@nso% configuration mode
```

Operational Mode

```
admin@nso# show devices list
NAME   ADDRESS   NED ID        ADMIN STATE
-----------------------------------------
P11 127.0.0.1 cisco-ios-xr unlocked  P12
127.0.0.1  cisco-ios-xr  unlocked    P21
127.0.0.1  cisco-ios-xr  unlocked   PE11
127.0.0.1   cisco-ios   unlocked    PE12
127.0.0.1   cisco-ios   unlocked    PE21
127.0.0.1  cisco-ios-xr  unlocked   PE22
127.0.0.1 cisco-ios-xr  unlocked
...
```

Configuration Mode

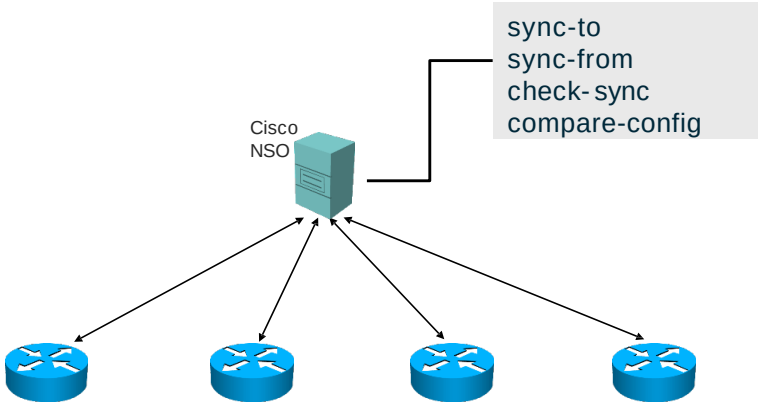
```
admin@nso# configure
admin@nso(config)# show full-configuration
devices device ce0

devices device PE11
 address   127.0.0.1
 port      10101
 ssh host-key ssh-dss
  key-data ...
 !
 ...
```

Device Configuration Management

Synchronizing **from** Device

- Device Configurations in NSO and actual Device Configuration should match
- After initial device discovery or import, it makes sense to synchronize configurations from devices

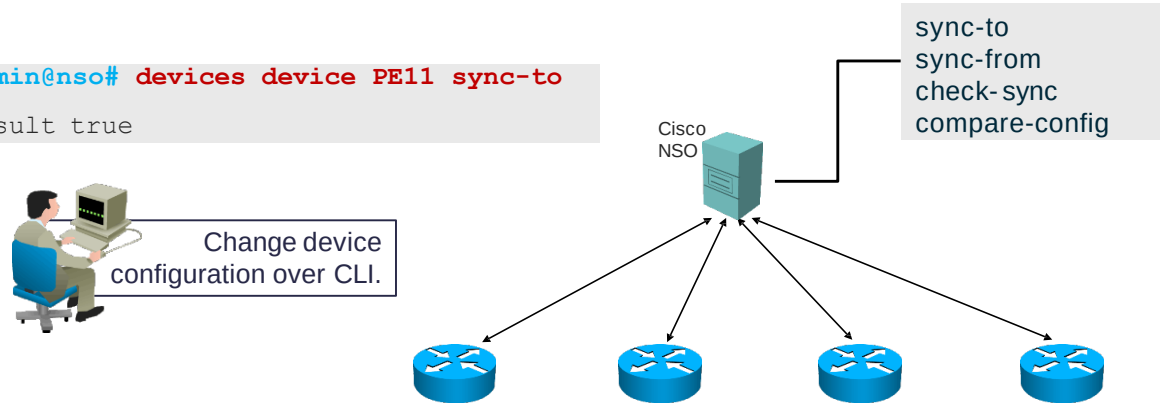
```
admin@nso# devices sync-from

  sync-result {
     device  lb0
     result
     true
     }
```



Synchronizing **to** Device

- When a device has been configured out of band
- Clears up rogue configuration
- “dry-run” option available to check changes

```
admin@nso# devices device PE11 sync-to
result true
```



Check Sync

- Check if a device has been configured out of band

```
admin@nso# devices check-sync
sync-result {
      device PE11
      result in-
      sync
}
...
```

- Check if a subset of managed devices has been configured out of band

```
admin@nso# devices device PE1..9 check-sync
devices device PE1 check-sync
result in-sync
devices device PE2 check-sync
result in-sync
devices device PE3 check-sync
```

Comparing Configuration

- Compare out-of-sync device configuration

```
admin@nso(config)# devices device PE11 check-sync
result out-of-sync
info got: 334bb33aae40155831edfa0b6a978f39 expected: a1424cd35da4499f6a71b3d38ae648a8

admin@nso(config)# devices device PE11 compare-config
diff
 devices {
     device PE11 {
         config {
             ios:interface {
                 Loopback 10 {
                     ip {
                         address {
                             primary {
-                                address 10.1.1.1;
+                                address 2.2.2.2;
                             }
                         }
                     }
```

"-" represents configuration items that should be deleted from the CDB in order to be the same as on the device

"+" represents configuration items that should be added to the CDB in order to be the same as on the device

Displaying Configuration

- Display only new parts of configuration:

```
admin@nso# config
admin@nso(config)# devices device PE11 config ios:interface Loopback 20
admin@nso(config-if)# ip address 10.2.2.2 255.255.255.255
admin@nso(config)# devices device PE11 config ios:interface Loopback 30
admin@nso(config-if)# ip address 10.3.3.3 255.255.255.255
admin@nso(config-if)# show configuration
devices device PE11
 config
  ios:interface Loopback30
   ip address 10.3.3.3 255.255.255.255
   no shutdown
  exit
 !
!
admin@nso(config)#
```

Displays current configuration items in the current configuration mode

Displaying Configuration (Cont.)

- Display only new parts of configuration:

```
admin@nso(config-if)# top
admin@nso(config)# show
configuration  devices device
PE11
 config
  ios:interface Loopback20
   ip address 10.1.1.1 255.255.255.255
   no
  shutdown
  exit
  ios:interface Loopback30
   ip address 10.3.3.3 255.255.255.255
   no shutdown
  exit
 !
!
```

Go to root of the data tree to display the all configuration items of the configuration session

Configuring Devices

- Configuration change happens after final commit statement

```
admin@nso# config
admin@nso(config)#
admin@nso(config)# devices device PE11 config ios:interface Loopback 20
admin@nso(config-if)# ip address 10.2.2.2 255.255.255.255
admin@nso(config)# devices device PE11 config ios:interface Loopback 30
admin@nso(config-if)# ip address 10.3.3.3 255.255.255.255
admin@nso(config-if)# commit
admin@nso#
```

Displaying Configuration

- Display entire CDB:

```
admin@nso# show running-config
  or
admin@nso(config)# show full-configuration
```

- Display portion of CDB:

```
admin@nso# show configuration devices device PE11 config ios:interface Loopback
devices device PE11  config
  ios:interface Loopback0
   ip address 10.1.1.1 255.255.255.255
   no shutdown
...
```

Rollbacks

- Every transaction has a corresponding rollback file:

```
admin@nso:~/ncs-run/logs$ ls rollback*
admin@nso:~/ncs-run/logs$ more rollback10157
```

Displays what NSO did

```
admin@nso# show configuration commit 10157
devices device PE11
 config
  no ios:interface Loopback10
 !
!
admin@nso# show configuration rollback 10157
!
! Created by: admin
! Date: 2016-01-14 14:40:58
! Client: cli
!
devices device PE11
 config
  ios:interface Loopback10
   ip address 10.1.1.1 255.255.255.255
   no shutdown
```

Displays what NSO would do if you execute a rollback

Rollbacks – Examples

- Rollback 3 latest transactions (last change ID is 10157):

```
admin@nso(config)# rollback configuration 10155
```

- Rollback only changes done in 3rd latest transaction:

```
admin@nso(config)# rollback selective 10155
```

- Rollback interface changes on PE11 in the 3 latest transactions:

```
admin@nso(config)# rollback configuration 10155 devices device PE11 config ios:interface
```

- Rollback interface changes on PE11 in the 3rd latest transaction:

```
admin@nso(config)# rollback selective 10155 devices device PE11 config ios:interface
```

Configured Devices

- List devices with basic parameters
  - Device address and description
  - NED ID
  - Administrative state

```
admin@nso# show devices list

-------------------------------------------------------
NAME   ADDRESS    DESCRIPTION  NED ID         ADMIN STATE

P11    127.0.0.1  -           cisco-ios-xr   unlocked
P12    127.0.0.1  -           cisco-ios-xr   unlocked
P21    127.0.0.1  -           cisco-ios-xr   unlocked
PE11   127.0.0.1  -           cisco-ios      unlocked
PE12   127.0.0.1  -           cisco-ios      unlocked
PE21   127.0.0.1  -           cisco-ios-xr   unlocked
PE22   127.0.0.1  -           cisco-ios-xr   unlocked
PE31   127.0.0.1  -           cisco-ios-xr   unlocked
PE41   127.0.0.1  -           cisco-ios-xr   unlocked
```

Device Connection Management

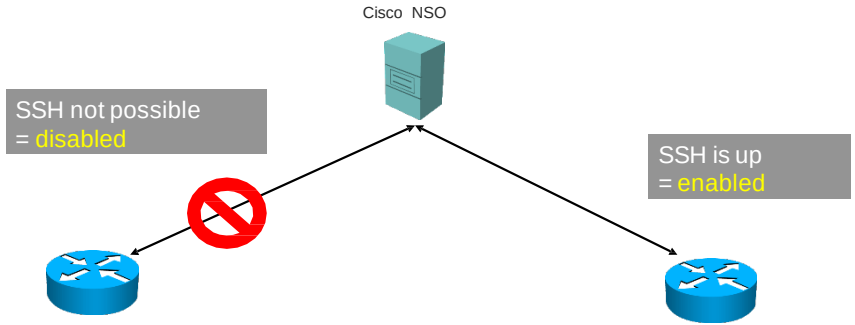
Device States

Oper State	Description
Enabled	Device is reachable from NSO.
Disabled	Device is unreachable from NSO.

Admin State	Description
Southbound-locked (default)	Configuration change is possible not pushed to the device.
Locked	Configuration change is not possible.
Unlocked	Configuration change is possible and pushed to the device.

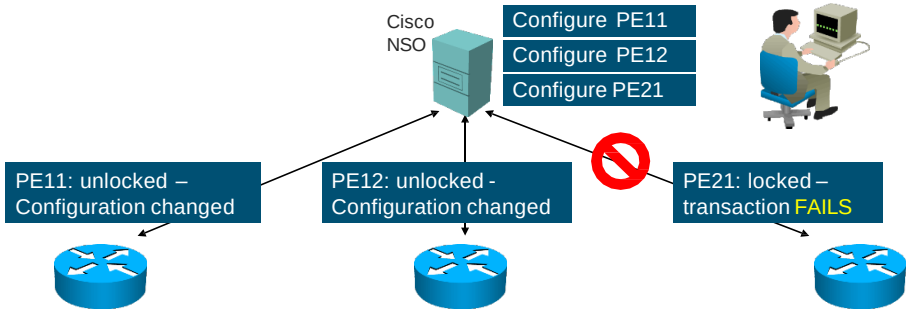
Oper State

- The actual state of a managed device
- It is either 'enabled' or 'disabled'
- If NSO initiates operations and the device is unreachable the oper state is set to disabled



Admin State

- Southbound-locked (default): possible to manipulate the configuration but changes are never pushed to the device
- Locked: all changes to device are forbidden
- Unlocked: this is the state we set a device into when the device is operational. All changes to the device will be sent southbound



Adding a Device

- Manually: useful for small number of devices (e.g. development and testing)
- Cloning: replicate a device from an existing device
- Templates: replicate a device from a template
- Bulk upload: useful for initial definition of many devices

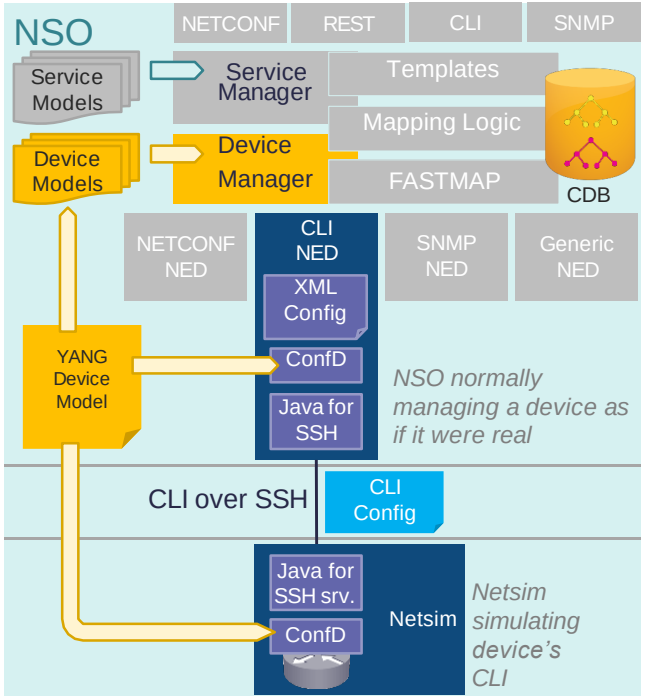
```
admin@nso(config)# devices device PE101 address 10.1.1.1
admin@nso(config-device-PE101)# device-type cli ned-id cisco-ios protocol ssh
admin@nso(config-device-PE101)# authgroup default
admin@nso(config-device-PE101)# commit
Commit complete.
```

Lab Exercise 1

Netsim

- Optional add-on to NEDs
- Simulates device's native CLI
- Useful for development and testing
- Lightweight: enables simulation of many devices on a desktop
- Uses YANG model and ConfD in reverse to simulate a device's CLI

- ! Netsim is not a replacement for real devices !



Starting Simulated Devices

- Below example creates 3 Cisco IOS devices:

```
cisco@ncs:~/lab$ ncs-netsim create-network <NED package> <#N devices>
cisco@ncs:~/lab$ ncs-netsim create-network $NCS_DIR/packages/neds/cisco-ios 3 c
```

Create 3 devices

Use prefix “c” for device names

- Simply run netsim inside the lab folder

```
cisco@ncs:~/lab$ ncs-netsim start
  DEVICE c0 OK STARTED
  DEVICE c1 OK STARTED
  DEVICE c2 OK STARTED
cisco@ncs:~/lab$
```

Netsim Options

- Netsim supports creating, adding and deleting simulated devices on the fly

```
cisco@ncs:~$ ncs-netsim --help

Usage ncs-netsim  [--dir <NetsimDir>]

                  create-network <NcsPackageDir> <NumDevices> <Prefix> |
                  add-to-network <NcsPackageDir> <NumDevices> <Prefix> |
                  delete-network                                       |
                  [-a | --async]  start [devname]    |
                  [-a | --async ] stop [devname] |
                  [-a | --async ] reset [devname] |
                  [-a | --async ] restart [devname] |
                  list                          |
                  is-alive [devname]           |
                  status [devname]             |
                  whichdir                      |
                  ncs-xml-init [devname]       |
                  packages                      |
                  netconf-console devname [XpathFilter] |
                  [-w | --window] [cli | cli-c | cli-i] devname
```

Access Simulated Devices

- You can run the CLI towards the simulated devices

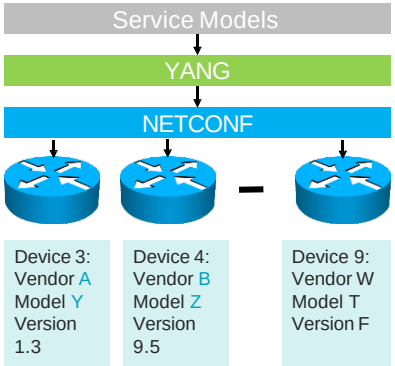
```
cisco@ncs:~$ ncs-netsim cli-i c1
admin connected from 127.0.0.1 using console *

c1> enable
c1# show running-config

class-map m
 match mpls experimental topmost 1
 match packet length max 255
 match packet length min 2
 match qos-group 1
!
c1# exit
```

Introduction to YANG

YANG

"YANG is a data modeling language used to model configuration and state data manipulated by the Network Configuration Protocol (NETCONF), NETCONF remote procedure calls, and NETCONF notifications. YANG is used to model the operations and content layers of NETCONF" IETF RFC 6020

The NETCONF protocol allows a manager to set configuration, query configuration and state and execute actions on the device (similar to SNMP).



The YANG models describe everything there is to …
- Configure
- Monitor
- Admin actions
- Notifications

…for each device type and version (similar to MIBs).

Note! NETCONF is not the only protocol. NSO also uses NEDs.

YANG Characteristics

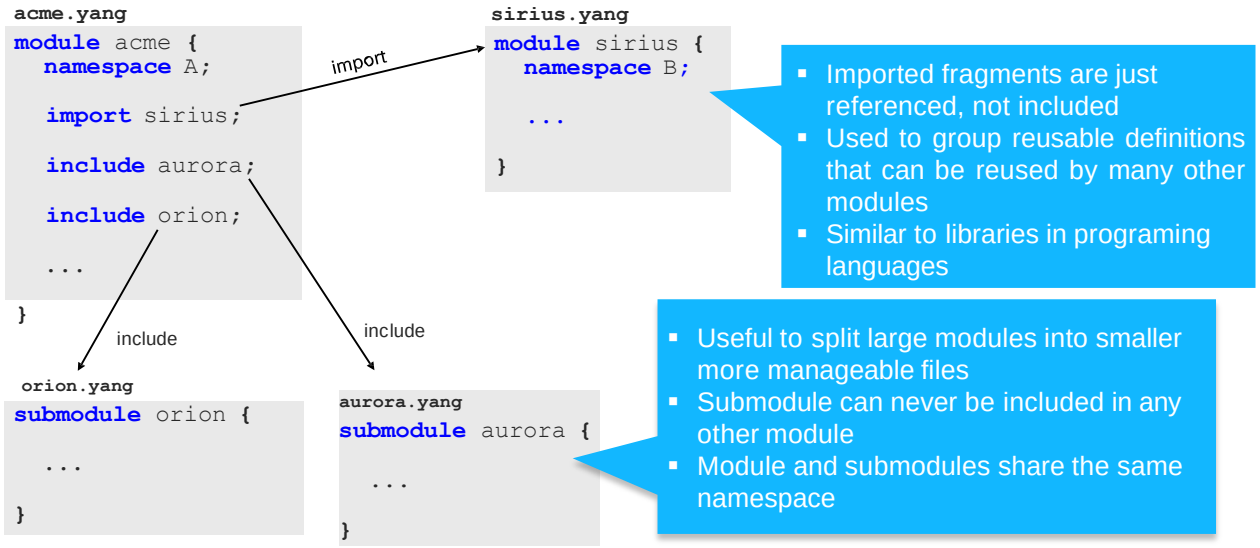
- Designed for **readability**
- **Simplicity** enables fast service model development
- **Modularity** and **hierarchy** enable reuse of code

Modeling a Football team in Yang

- Team should have a name.
- Has multiple players.
  - Players have names.
  - They have specific positions.
  - They have an age.

```
container FootballTeam {
 leaf TeamName {
   type string;
 }
 list Player {
  leaf PlayerName {
   type string;
  }
  leaf Position {
   type string;
  }
  leaf Age {
   type uint8;
  }
 }
}
```

YANG Model Hierarchy and Modularity



- Imported fragments are just referenced, not included
- Used to group reusable definitions that can be reused by many other modules
- Similar to libraries in programing languages

- Useful to split large modules into smaller more manageable files
- Submodule can never be included in any other module
- Module and submodules share the same namespace

Yang Data Types

YANG Data Types

- Built-in data types
- Derived data types:
    - RFC-defined data types (RFC 6991)
    - NSO-defined data types
    - Custom data types

YANG Supports a Number of Data Types

Built-in Types

Name	Description
int8/16/32/64	Integer
uint8/16/32/64	Unsigned integer
decimal64	Non-integer
string	Unicode string
enumeration	Set of alternatives
boolean	True or false
bits	Boolean array
binary	Binary BLOB
leafref	Reference
identityref	Unique identity
empty	No value, void
union	Choice of member types
instance-identifier	References a data tree node

Derived Types

```
typedef my-base-int32-type {
  type int32 {
    range "1..4 | 10..20";
  }
}

typedef derived-int32 {
  type my-base-int32-type {
    range "11..max";
  }
}
```

```
typedef string255 {
  type string {
    length "1..255";
  }
}

typedef derived-str {
  type string255 {
    length "11 | 42..max";
    pattern "[0-9a-fA-F]*";
  }
}
```

Derived Type Restrictions

- Range
  - Applies to integer and decimal types
    - "x .. y"
    - "min .. x" , "x .. max".
- Length
  - For string type
  - Same syntax
- Pattern
  - For string type
  - Uses regular expressions
- Fraction-digits
  - Applies to decimal type

```
typedef acl-num-type {
  type int32 {
    range "1..199 | 1300..2699";
  }
}
typedef std-acl-num-type {
  type acl-num-type {
    range "min..99 | 1300..1999";
  }
}
typedef ext-acl-num-type {
  type acl-num-type {
    range "100..199 | 2000..max";
  }
}
typedef acl-name-type {
  type string {
    length "1..50";
    pattern "[a-zA-Z][a-zA-Z0-9-_]*";
  }
}
typedef std-acl-name-type { type acl-name-type; }
typedef ext-acl-name-type { type acl-name-type; }
typedef const-type {
  type decimal64 {
    fraction-digits 4;
    range "2.7182 .. 3.1415";
  }
}
```

cisco Live!

Combined Derived Types

- Enumeration
  - Defines a set of names
- Union
  - Used to define a type with a combination of two or more other types

```
typedef intf-ip-addr-type {
  type union {
    type inet:ip-address;
    type enumeration {
      enum "none";
      enum "unnumbered";
      enum "dhcp";
    }
  }
}

typedef acl-id-type {
  type union {
    type acl-num-type;
    type acl-name-type;
  }
}
```

YANG Types Example

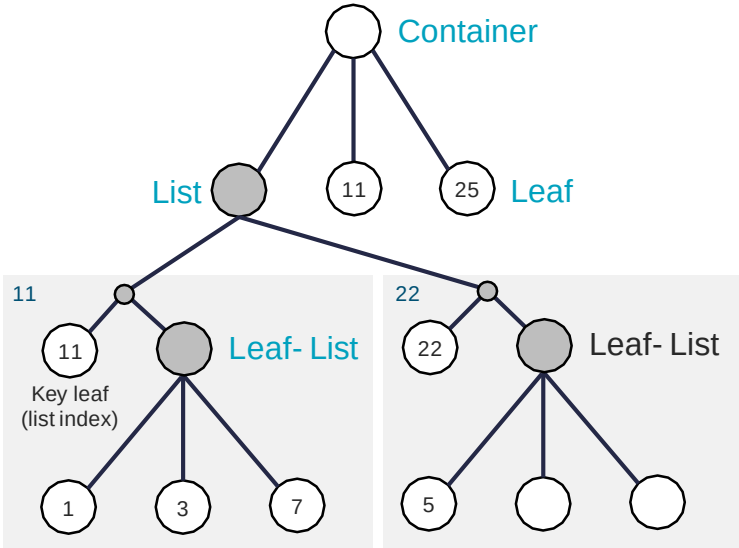

```
// percentage type
typedef percentage-type
{
  type uint8 {
    range
    "1..100";
  }
}
// Weekday type
typedef weekday-type
  {  type
  enumeration {
    enum Mon;
    enum Tue;
    enum Wed;
    enum Thu;
    enum Fri;
    enum Sat;
    enum Sun;
  }
}
// Hour & minute & optional second
type  typedef hhmm-type {
  type string {
    pattern '([0-1]?[0-9]|2[0-3]):' +
            '([0-5][0-9])(:[0-5][0-9])?';
  }
}
// Route Distinguisher AS:NUM or
IP:NUM  typedef rd-type {
  type string {
    pattern '((\d+)((\.\d+){3})?)\:\d+';
  }
}
```

```
// DSCP type
typedef dscp-type;
  type union;
    type uint8 { range "0..63";
    }  type enumeration {
      enum af11;
      enum af12;
      enum af13;
      enum af21;
      enum af22;
      enum af23;
      enum af31;
      enum af32;
      enum af33;
      enum af41;
      enum af42;
      enum af43;
      enum cs1;
      enum cs2;
      enum cs3;
      enum cs4;
      enum cs5;
      enum cs6;
      enum cs7;
      enum default;
      enum dscp;
      enum ef;
      enum precedence;
    }
  }
}
```

YANG Data Definitions

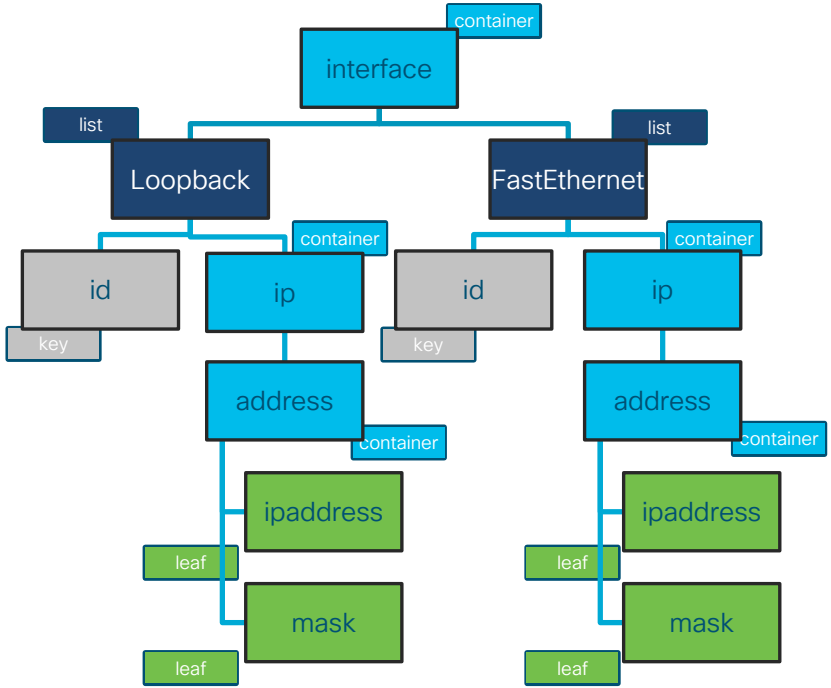
Basic YANG Statements Defining Data Nodes

YANG	Programming Equivalent	Description
Leaf	Variable	Contains a single value of a specific type
Leaf-List	Array	Contains a list of values of the same type
Container	Record	Contains a single structure containing zero or more values or other statements (hierarchy)
List	Array of Records	Contains a list of zero or more sets of values and other statements (hierarchy)



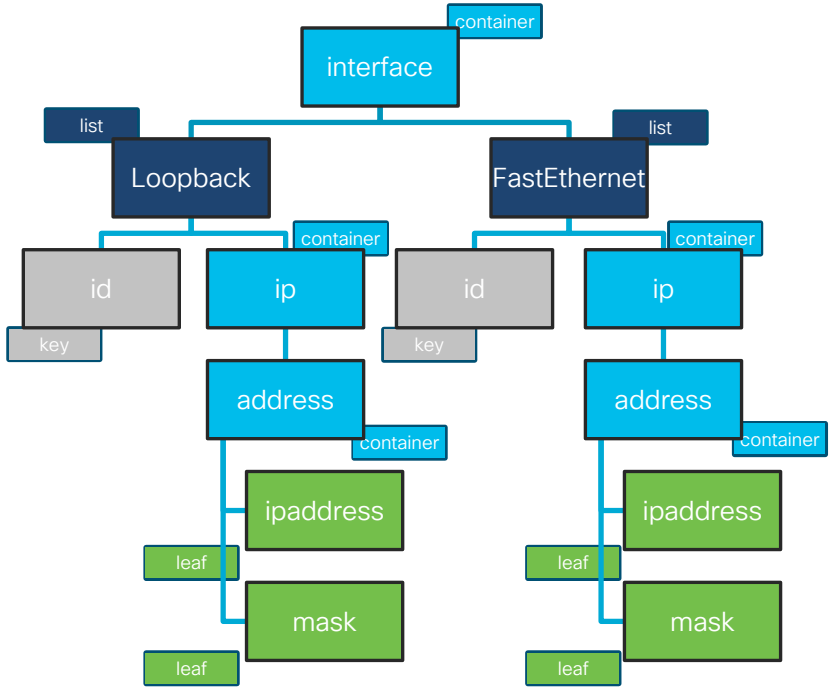
Modeling Cisco IOS commands in Yang

```
interface Loopback 1
        ip address 10.0.0.1 255.255.255.255
interface FastEthernet 1/2
        ip address 10.1.0.2 255.255.255.0
```

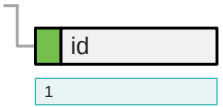


Modeling Cisco IOS commands in Yang (Cont)

```
container interface {
       list Loopback {
              key id;
              leaf id {
                type uint16 {
                 range "1..65535";
                 }
              }
              container ip {
                     container address {
                            leaf ipaddress {
                                   type string;
                            }
                            leaf mask {
                                   type string;
                            }

                     }
              }
       }
       list FastEthernet {
              key id;
              leaf id {
                     type string;
              }
              container ip {
                     container address {
                            leaf ipaddress {
                                   type string;
                            }
                            leaf mask {
                                   type string;
                            }

                     }
              }
       }
}
```



Leaf



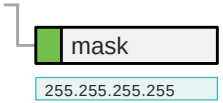
XPath:
/interface/Loopback [id="1"]

YANG (data model)

```
leaf id {
       type uint16 {
         range "1..65535";
       }
}
```

XML (data)

```
<interface>
 <Loopback>
   <id>1</id>
 </Loopback>
</interface>
```



XPath:
/interface/Loopback[id='1']/ip/address/mask

YANG (data model)

```
leaf mask{
       type string;
}
```

XML (data)

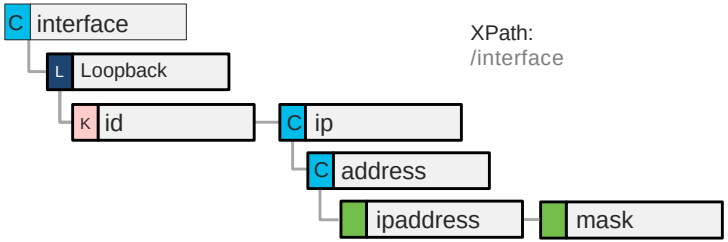
```
<interface>
  <Loopback>
   <id>1</id>
   <ip>
    <address>
     <ipaddress></ipaddress>
     <mask>255.255.255.255</mask>
    </address>
   </ip>
  </Loopback>
</interface>
```

- Single value using a built-in or derived data type
- Zero or one instance

Leaf Attributes

Attribute	Description
config	Whether this leaf is a configurable value ("true") or operational value ("false"). Inherited from parent container if not specified
default	Specifies default value for this leaf. Implies that leaf is optional
mandatory	Whether the leaf is mandatory ("true") or optional ("false")
must	XPath constraint that will be enforced for this leaf
type	The data type (and range etc) of this leaf
when	Conditional leaf, only present if XPath expression is true
description	Human readable definition and help text for this leaf
reference	Human readable reference to some other element or spec
units	Human readable unit specification (e.g. Hz, MB/s, °F)
status	Whether this leaf is "current", "deprecated" or "obsolete"

Container



- Used to group one or more other statements
- Has no data type by itself
- May have an implicit meaning

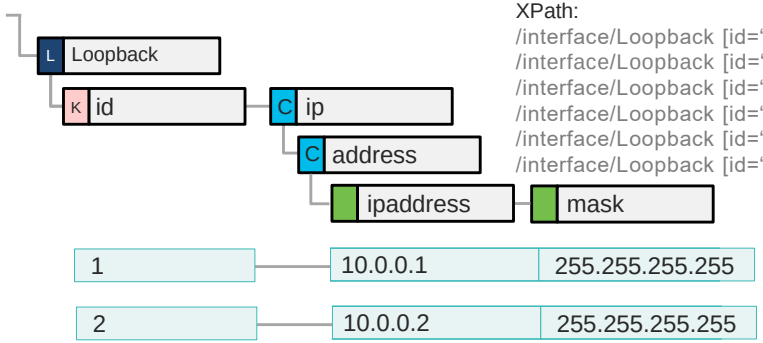
YANG (data model)

```
container interface {
        list Loopback {
                key id;
                leaf id {}
                container ip {
                        container address {
                                leaf ipaddress {}
                                leaf mask {}
                        }
                }
        }
}
```

XML (data)

```
<interface>
</interface>
```

List



XPath:
/interface/Loopback [id="1"]
/interface/Loopback [id="1"]/ip / address / ipaddress
/interface/Loopback [id="1"]/ip / address / mask
/interface/Loopback [id="2"]
/interface/Loopback [id="2"]/ip / address / ipaddress
/interface/Loopback [id="2"]/ip / address / mask

YANG (data model)

```
list Loopback {
        key id;
        leaf id {
                type uint16 {
                        range "1..65535";
                }
        }
        container ip {
                container address {
                        leaf ipaddress {
                                type string;
                        }
                        leaf mask {
                                type string;
                        }

                }
        }
}
```

- Contains one or more sub statements
- Requires one unique identifier (key)
- Zero or more instances

XML (data)

```
<interface>
  <Loopback>
   <id>1</id>
     …
  </Loopback>
   <Loopback>
   <id>2</id>
     …
  </Loopback>
</interface>
```

List: Single Key

yang	1010	Yan Goode	admin
hawk	1152	Ron Hawk	oper
ling	1202	Lin Grossman	viewer

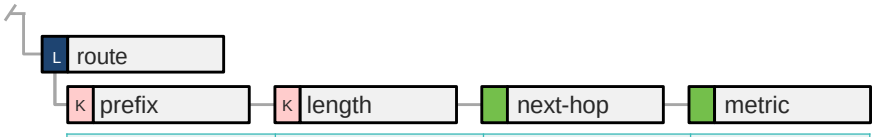
XPath:

```
/ user [name='yang'] / name      =  yang
/ user [name='yang'] / uid       =  1010
/ user [name='yang'] / full-name =  Yan Goode
/ user [name='yang'] / class     =  admin
```

```
list user {
  key name;
  ...
}
```

- The key refers to a leaf
- The key values must be unique

List: Combined Key



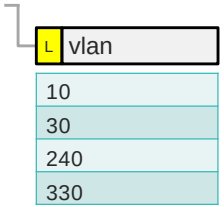
192.0.2.0	24	10.100.1.2	20
198.51.100.0	24	192.168.31.5	40
203.0.113.0	24	172.31.11.1	50

XPath:
/ route [prefix='198.51.100.0'][length='24'] / next-hop = 192.168.31.5
/ route [prefix='198.51.100.0'][length='24'] / metric = 40

```
list route {
  key "prefix length";
  ...
}
```

- Keys refer to leaf statements
- Key combinations must be unique

Leaf-list



XPath:
vlan

YANG (data model)

```
leaf-list vlan {
  type uint16 {
    range "0..2147483647";
  }
}
```

XML (data)

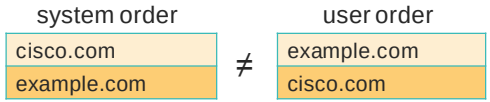
```
<vlan>10</vlan>
<vlan>20</vlan>
<vlan>240</vlan>
<vlan>330</vlank>
```

- Example: `switchport trunk allowed vlan 10,30,240,330`
- Unique values using a single built-in or derived data type
- Zero or more instances

List and Leaf-List Attributes

Attribute	Description
max-elements	Max number of elements in a list
min-elements	Min number of elements in a list
ordered-by	List entries are sorted by "system" or "user". System means elements are sorted in a natural order (numerically, alphabetically, etc). User means the order the operator entered them in is preserved.

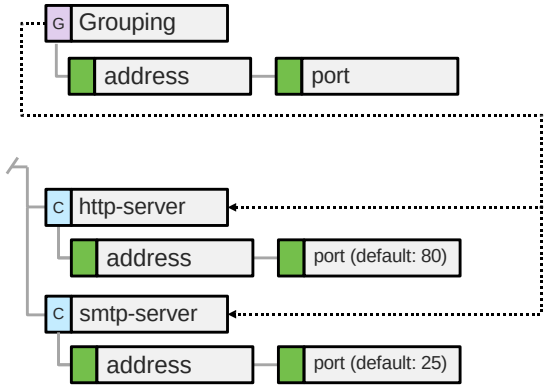
```
leaf-list domain-search {
  type string;
  ordered-by user;
  min-elements 1;
  max-elements 2;
}
```



Grouping

YANG (data model)

```
grouping app {
  leaf address {
    type inet:ip-address;
  }
  leaf port {
    type inet:port-number;
  }
}

container http-server {
  uses app {
    refine port {
      default 80;
    }
  }
}

container smtp-server {
  uses app {
    refine port {
      default 25;
    }
  }
}
```

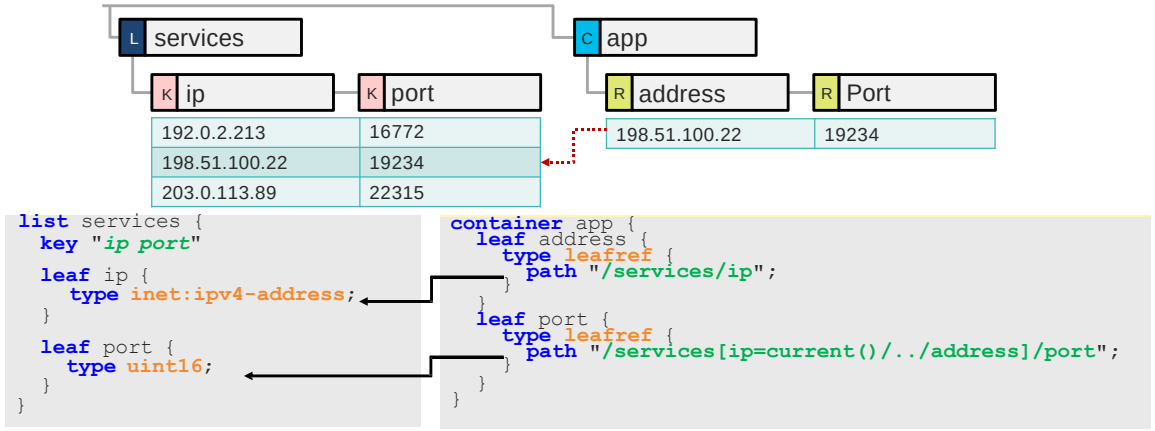


10.1.1.1	80
10.1.2.1	25

XPath:
/ http-server
/ http-server / address
/ http-server / port
/ smtp-server
/ smtp-server / address
/ smtp-server / port

- Used to group YANG code for reuse
- Can be refined upon use

Leafref



- Refer to a leaf or leaf-list elsewhere in the tree
- Inherits data type from referenced node
- The path keyword is used to match:

  the position in the tree

  the key(s) if referring to a leaf-list or leafs within a list (all key leafs must be referenced)

YANG Module Structure

acme.yang

```
module acme {
  namespace "http://acme.example.com/module";
  prefix acme;
  organization "ACME Inc.";
  description "Module describing the ACME products";

  revision 2007-06-09 { description "Initial revision."; }

  import ietf-yang-types {  prefix yang; }
  include acme-system;

  typedef percentage-type {
    type uint8 {     range "1..100";    }
  }

  grouping ifdata {
    leaf ipv4-address {
      type inet:ipv4-address;
    }
    leaf ipv4-mask {
      type inet:ipv4-address;
    }
  }
```

Header Information

Revision Information

Imports & Includes

Type Definitions

Reusable Node Declarations

YANG Module Structure(Cont.)

```
  list loopback-interface {
    key "Loopback";
    leaf Loopback {
      type string;
    }
    uses ifdata;
  }

  notification link-failure {
    description "A link failure has been detected";
    leaf if-name          { type string; }
    leaf if-admin-status  { type uint8;  }
  }

  rpc clear-interface {
    input {
      leaf interface-name { type string; }
    }
    output {
      leaf status { type string; }
    }
  }
}
```

Configuration & Operational Data Declarations

Action (RPC) & Notification Declarations

End of Module File

NSO and YANG

- NSO has it‘s own YANG models: tailf-ncs-*.yang
- The NSO YANG model can be augmented
- *tailf-ncs-devices.yang* references a flat list of devices
- A device is identified by a free form text string

```
container devices {
  tailf:info "The managed devices and device communication
  settings";  uses connect-grouping {
    augment "connect/input" {
      leaf-list device {
        tailf:info "Only connect to these
        devices.";  type leafref {  path
        '/devices/device/name';  }
      }
    }
  }
}
```

Devices and Device YANG Models

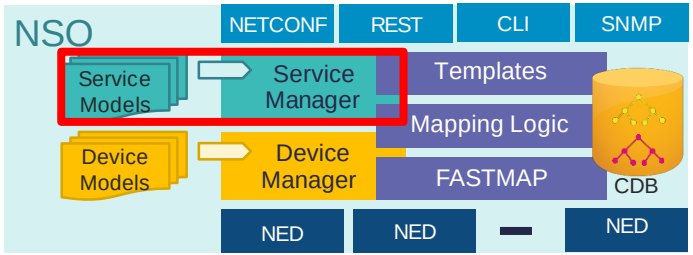
- To manage a device, we need a corresponding YANG model
- Device models are deployed as part of NED packages
- All imported modules appear under /devices/device/config in the CDB

The bridge between Network Automation and Business

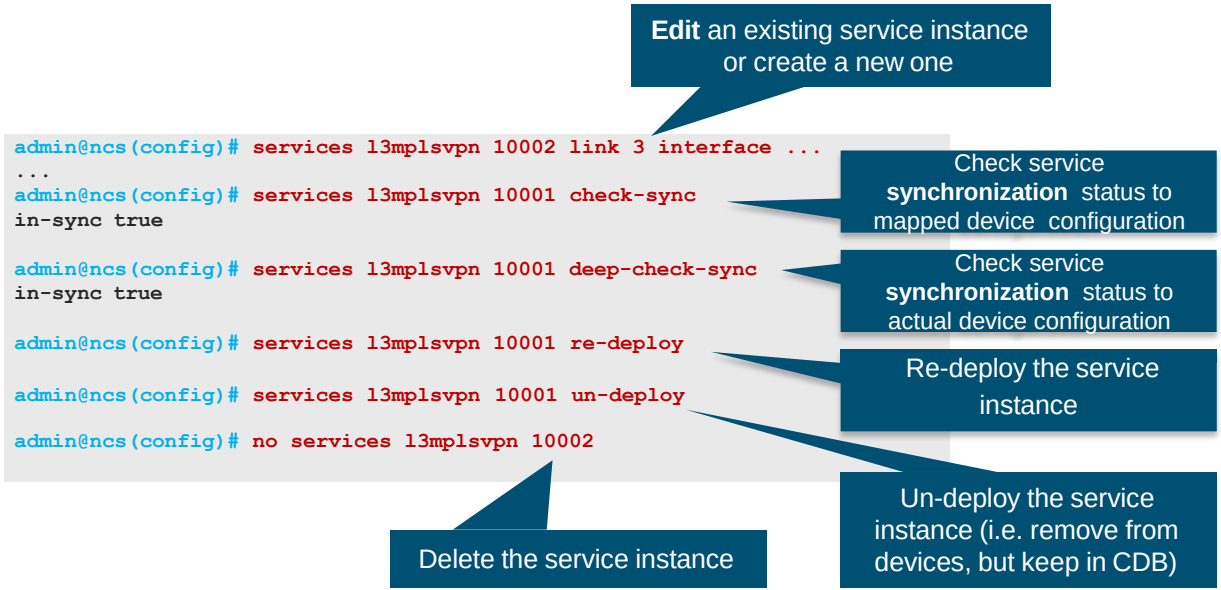
Service Manager

Service Manager

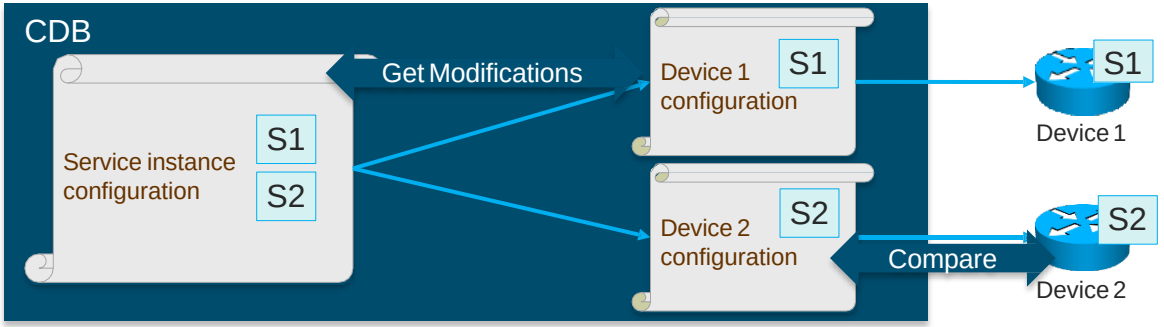
- Service modeling using YANG
- Service to Device mapping through Template + Python/Java code
- Service activation, modification, decommissioning, restoration
- Service synchronization with device configurations
- Aggregated operational data



NSO Service through CLI (Cisco Style)



Comparing Configurations: Service or Device

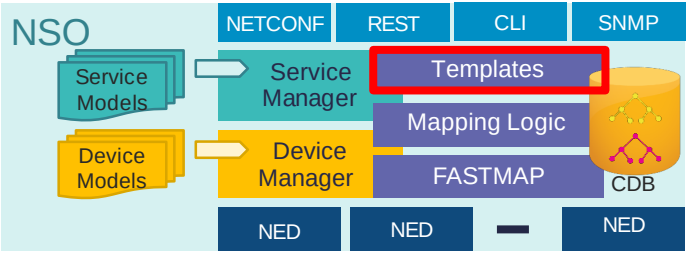



```
admin@nso# service l3mplsvpn S1 get-modifications
admin@nso# devices device PE11 compare-config
```

- Get device modifications made by a service instance
- Get discrepancies between CDB version of device configuration and actual device configuration

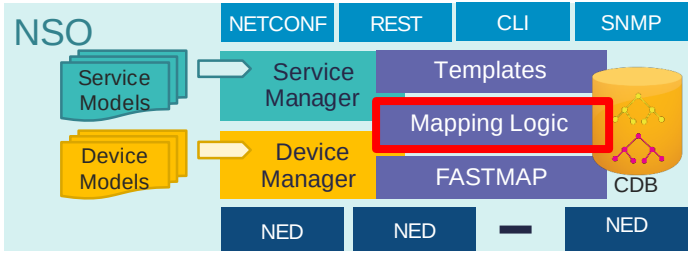
Templates

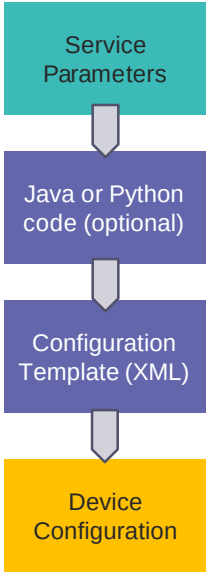
- Simplify changes in network configuration
- Two types:
  - Device templates for direct activation (e.g. through NSO CLI, REST)
  - Configuration templates for service mapping to device configurations



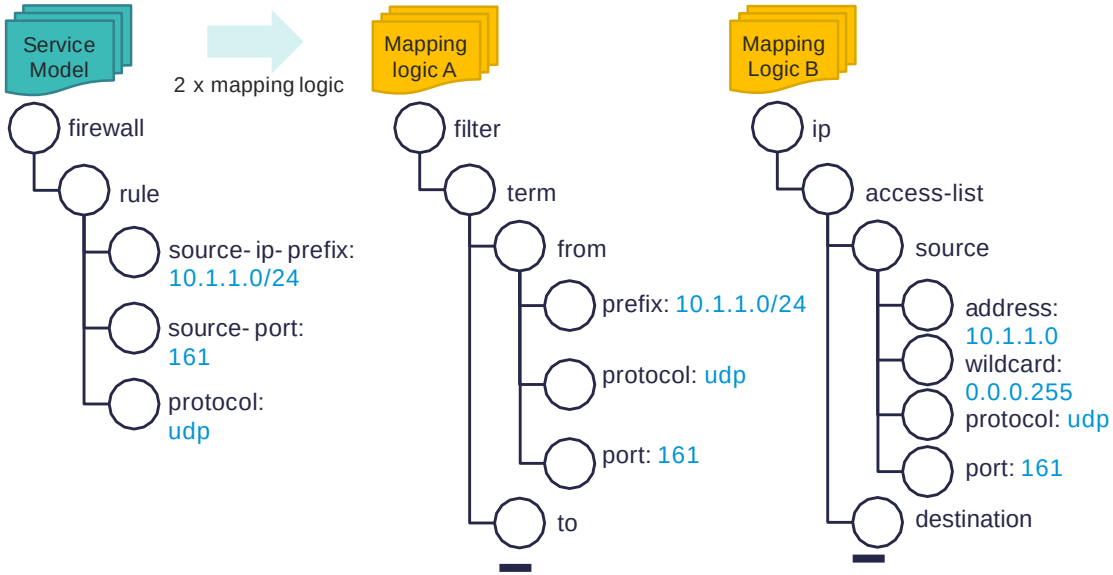
Mapping Logic

- Transforms the service models to device models
- When templates are not enough:
  - External call-outs can be made
  - Expressions or algorithms in Java or Python code can be used
- Uses FASTMAP
  - 1 page of code for a complex service





Mapping Logic (Cont.)



FastMap

FASTMAP

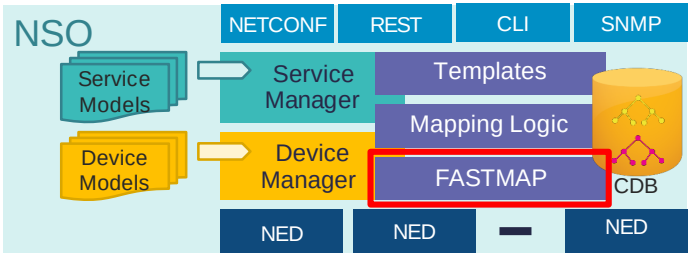
- FASTMAP controls and optimizes southbound configuration actions:

  Create: defines the “to-be” state

  Modify: FASTMAP algorithm calculates the necessary changes to be applied

  Delete: FASTMAP algorithm stores the “undo” operation needed to revert to the original state

- Operator is only concerned with defining the “to-be” state (i.e. create operation)



Example: Create a New Service Instance

- YANG service model governs service parameter API

```
admin@nso# config
Entering configuration mode terminal
admin@ncs(config)# services l2vpn ACME
admin@ncs(config-l2vpn-ACME)# pw-id 102

admin@ncs(config-l2vpn-ACME)# link Site1
admin@ncs(config-link-PE11)# device PE11
admin@ncs(config-link-PE11)# intf 0/5
admin@ncs(config-link-PE11)# remote 10.1.1.15
admin@ncs(config-link-PE11)# exit

admin@ncs(config-l2vpn-ACME)# link Site2
admin@ncs(config-link-PE11)# device PE12
admin@ncs(config-link-PE11)# intf 0/9
admin@ncs(config-link-PE12)# remote 10.1.1.11
admin@ncs(config-link-PE12)# top
admin@ncs(config)#
```

Example: Create a New Service

```
admin@ncs(config)# commit dry-run
device PE11
   config {
       ios:interface {
           GigabitEthernet 0/5 {
               xconnect {
  +                address 10.1.1.15;
  +                vcid 102;
  +                encapsulation mpls;
               }
           }
   }   }
device PE12
    config {
        ios:interface {
            GigabitEthernet 0/9 {
                xconnect {
  +                 address 10.1.1.11;
  +                 vcid 102;
  +                 encapsulation mpls;
                }
            }
        }
```

Service parameters mapped to device configuration

- NED device model governs device configuration parameters
- XML, Java or Python map service parameters to device configurations

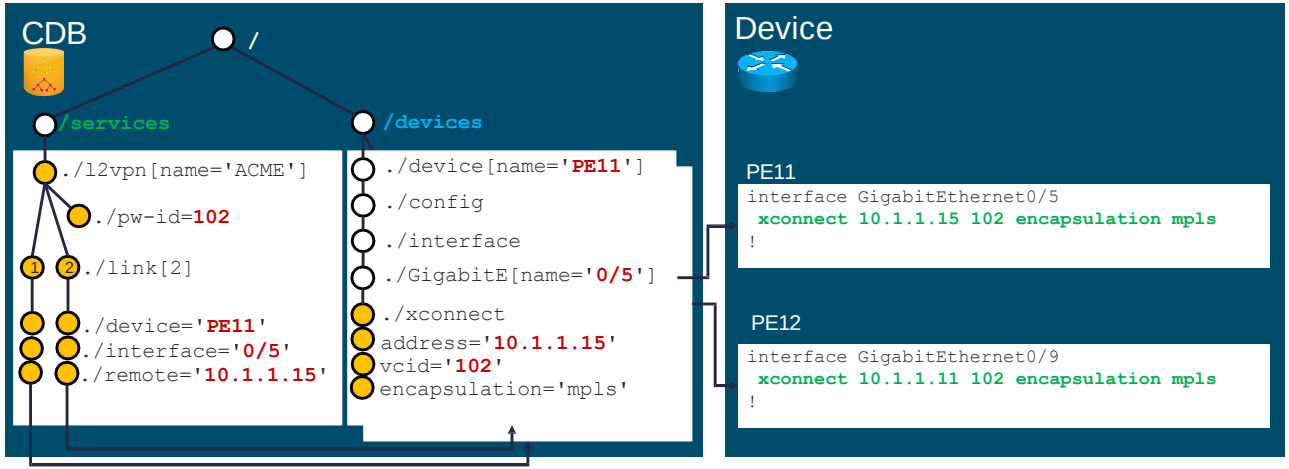
```
admin@ncs(config)# commit dry-run outformat native
device PE11
  interface GigabitEthernet0/5
   xconnect 10.1.1.12 102 encapsulation mpls
   exit
  exit
device PE12
  interface GigabitEthernet0/9
   xconnect 10.1.1.11 102 encapsulation mpls
   exit
  exit

admin@ncs(config)#
```

Can display native commands in case of CLI NED

- CLI NED's YANG model also contains additional information that governs mapping of YANG statements to CLI commands

cisco Live!

FASTMAP: Service Create



: service mapping & diff & remember reverse diff

Configuration sent to devices

Modify and Existing Service: Example 1

```
admin@nso# config
Entering configuration mode terminal
admin@ncs(config)# services l2vpn ACME
admin@ncs(config-l2vpn-ACME)# pw-id 122
admin@ncs(config-link-PE12)# top
admin@ncs(config)# commit dry-run
device PE11
   config {
       ios:interface {
           GigabitEthernet 0/5 {
               xconnect {
-                vcid 102;
+                vcid 122;
               }
           }
       }
   }
device PE12
   config {
       ios:interface {
           GigabitEthernet 0/9 {
               xconnect {
-                vcid 102;
+                vcid 122;
               }
admin@ncs(config)# commit dry-run outformat native
device PE11
  interface GigabitEthernet0/5
   xconnect 10.1.1.12 122_encapsulation mpls
   exit
  exit
device PE12
  interface GigabitEthernet0/9
   xconnect 10.1.1.11 122_encapsulation mpls
   exit
  exit
```

- Only one parameter changed
- CDB shows only minimal diff changes
- Changed parameter **affects both devices**

Modify and Existing Service: Example 2

```
admin@nso# config
Entering configuration mode terminal
admin@ncs(config)# services l2vpn ACME
admin@ncs(config-l2vpn-ACME)# link Site1
admin@ncs(config-l2vpn-ACME)# remote 10.1.1.12
admin@ncs(config-link-PE12)# top
admin@ncs(config)# commit dry-run
device PE11
   config {
       ios:interface {
           GigabitEthernet 0/11 {
               xconnect {
-                address 10.1.1.15;
+                address 10.1.1.12;
               }
           }
       }
   }
admin@ncs(config)# commit dry-run outformat native
device PE11
  interface GigabitEthernet0/5
   xconnect 10.1.1.12 122 encapsulation mpls
   exit
  exit
```

- Only one parameter changed
- CDB shows only minimal diff changes
- Changed parameter affects **only one device**

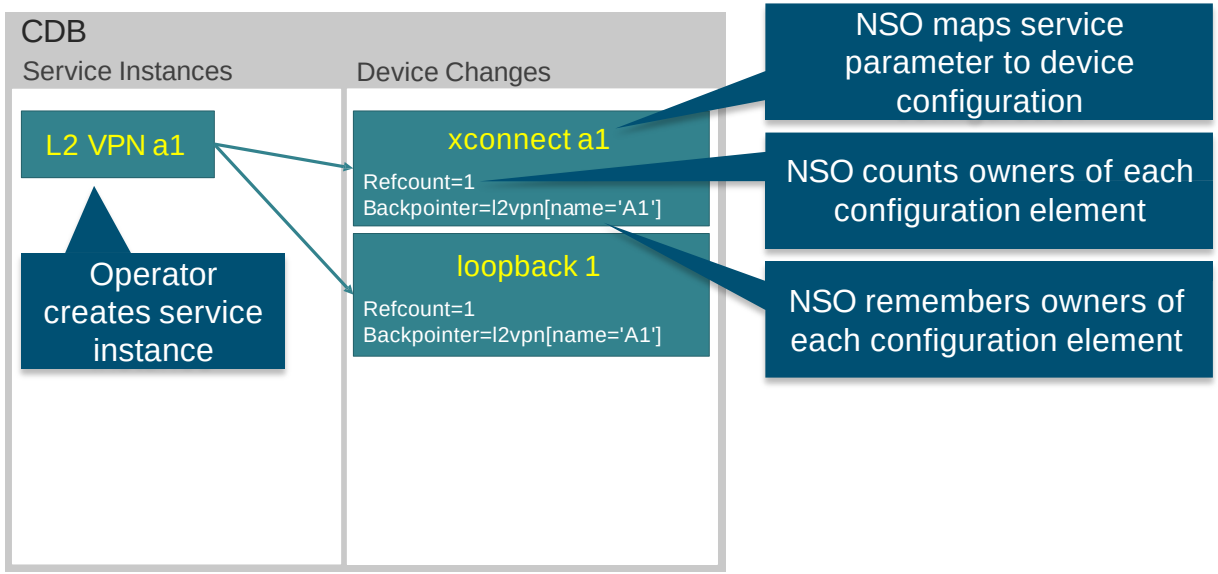
Delete a Service Instance

```
admin@nso# config
Entering configuration mode terminal
admin@ncs(config)# no services l2vpn ACME
admin@ncs(config)# commit dry-run
device PE11
   config {
        ios:interface {
            GigabitEthernet 0/6 {
                xconnect {
  -                 address 10.1.1.12;
  -                 vcid 122;
  -                 encapsulation mpls;
                }
            }
        }
   }
device PE12
   config {
       ios:interface {
           GigabitEthernet 0/9 {
               xconnect {
  -                address 10.1.1.11;
  -                vcid 122;
  -                encapsulation mpls;
               }
           }
       }
admin@ncs(config)# commit dry-run outformat native
device PE11
  interface GigabitEthernet0/6
   no xconnect 10.1.1.12 122 encapsulation mpls
   exit
device PE12
  interface GigabitEthernet0/9
   no xconnect 10.1.1.11 122 encapsulation mpls
   exit
  exit
```

- Only one parameter changed
- CDB shows only minimal diff changes
- Reverse diff used to undo the configuration changes of the service instance

Handling Services Sharing Resources

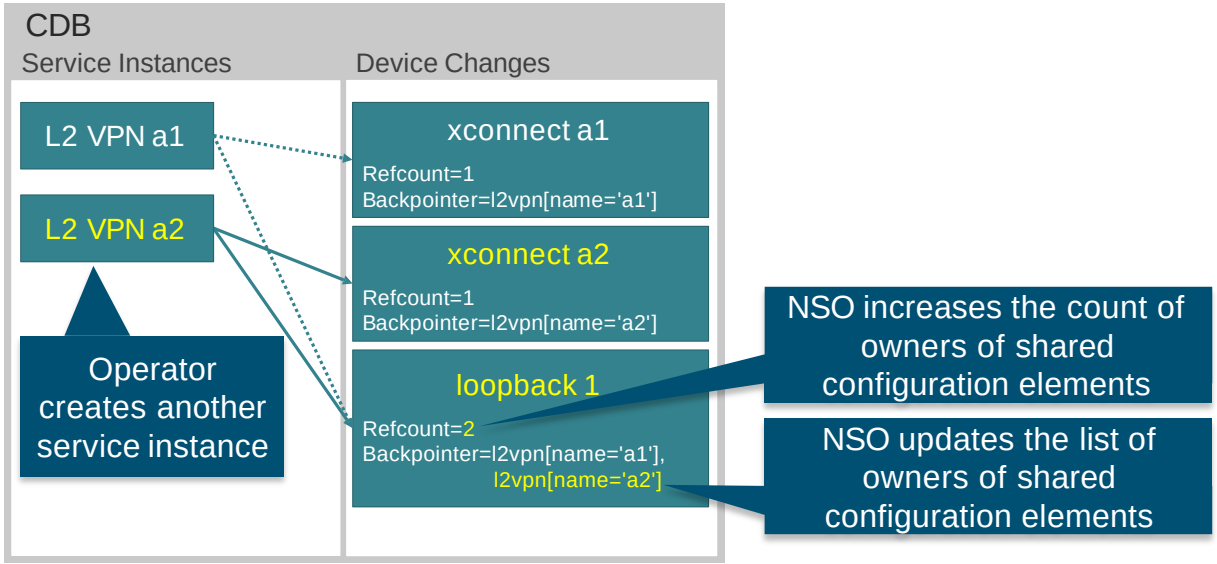
- Multiple service instances configuring the same shared prerequisite configuration



- Example: L2VPN service using common Loopback interfaces

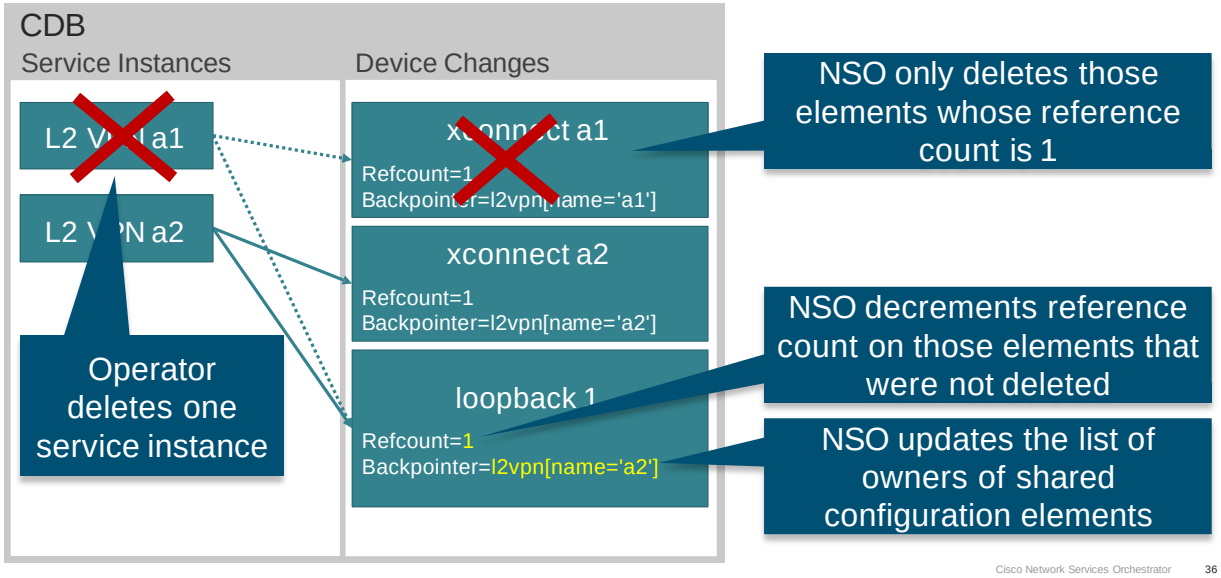
Handling Services Sharing Resources

- Second L2VPN service uses the same Loopback interface



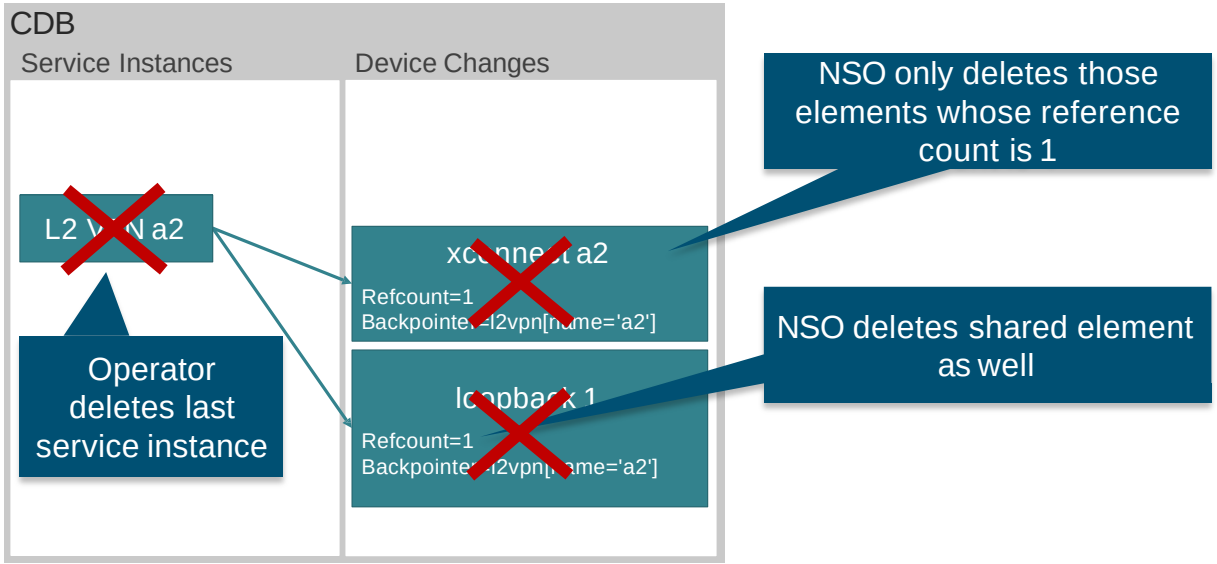
Handling Services Sharing Resources

- Deleting first L2VPN does not delete Loopback interface configuration



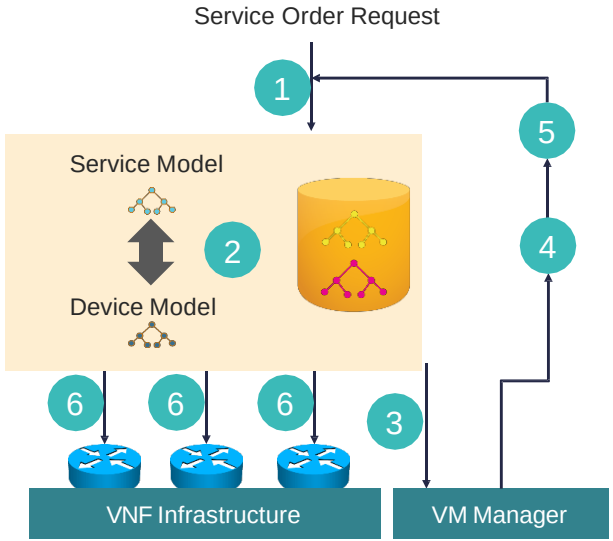
Handling Services Sharing Resources

- Deleting last L2VPN does delete Loopback interface configuration



Reactive FASTMAP
Example: Virtual Network Function (VNF)

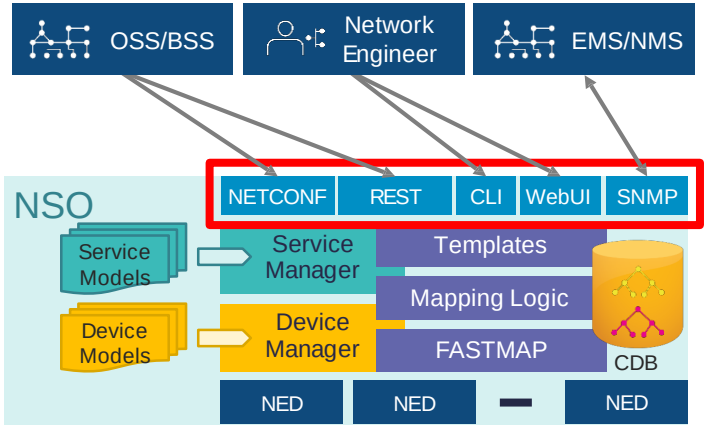
1. Service Request arrives
2. Service instance is created, but it requires VM resources first
3. Device Manager instruct VM Manager to spins up VMs
4. Once VM is ready a service trigger application invokes service again
5. Service redeploy is issued
6. Now devices can be configured



Northbound API and Integration options

Northbound APIs

- Human interaction:
  - NSO CLI
  - WebUI
- Automation integration:
  - NETCONF
  - REST/RESTCONF
  - JSON/RPC
- Monitoring:
  - SNMP
  - NETCONF



Human Interaction
Cisco NSO CLI Overview

- Single point of configuration
- Usability characteristics : Auto-completion and syntax help
- Two CLI modes : Operational mode and Configuration mode

```
admin@ncs# ?
Possible completions:
  alarms           - Alarm management
  autowizard       - Automatically query for mandatory elements
  cd               - Change working directory
  clear            - Clear parameter
  cluster          - Cluster configuration
  compare          - Compare running configuration to another configuration or a file
  complete-on-space - Enable/disable completion on space
  compliance       - Compliance reporting
  config           - Manipulate software configuration information
  describe         - Display transparent command information
  devices          - The managed devices and device communication settings
  display-level    - Configure show command display level
...
```

Human Interaction
Cisco NSO CLI for Configuration

Cisco Style

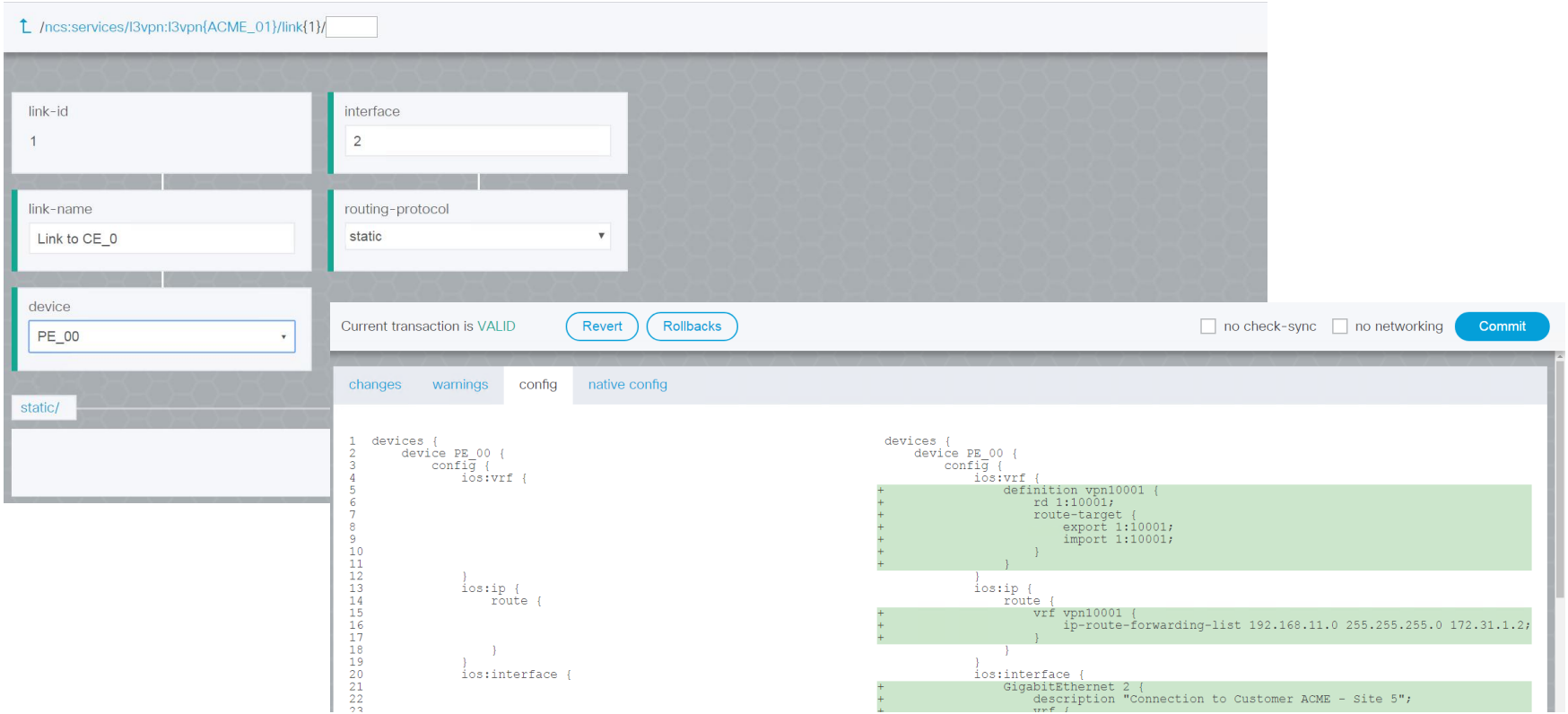
```
admin@nso#
admin@nso# config
admin@nso(config)#
admin@nso(config)# devices device PE11
...
admin@nso(config-device-PE11)# top
admin@nso(config)# exit | commit | abort
admin@nso#
admin@nso# show running-config
cluster authgroup default
 default-map same-user
 default-map same-pass
 umap admin
  same-user
...
```

Juniper Style

```
admin@nso> configure
Entering configuration mode
private  [ok][2016-01-27 11:14:12]

[edit]
admin@nso% set devices device PE11
...  admin@nso% exit | commit | revert
[ok][2016-01-27 11:15:44]
admin@nso> show configuration
cluster {
    authgroup default {
        default-map {
            same-user;
            same-pass;
        }
...
```

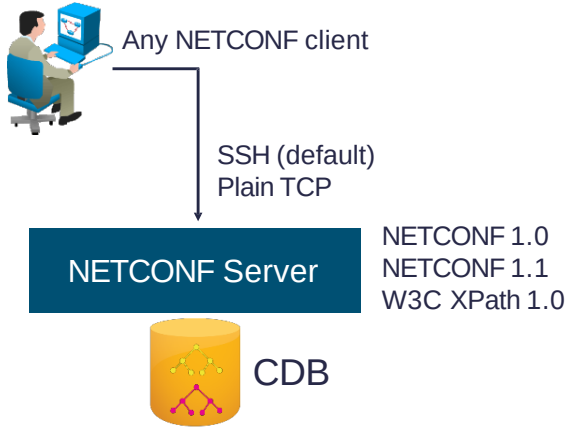
- Two modes: operational and configuration
- Two CLI styles: Cisco and Juniper (use the switch cli command to switch between them)

Human Interaction
Graphical User Interface



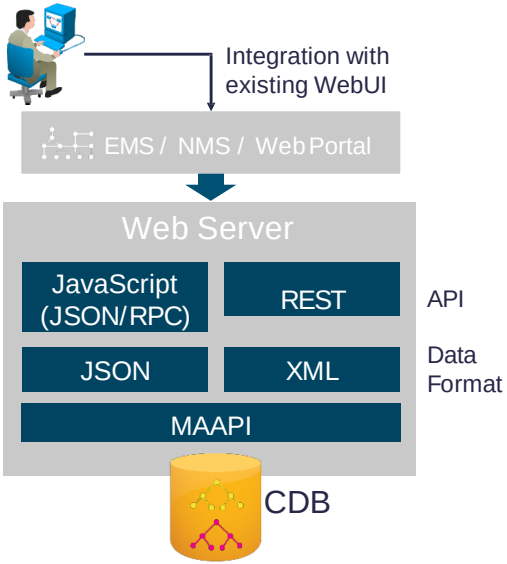
NSO Automation Integration
Cisco NSO NETCONF API

- Use cases:
  - Integration with other OSS/NMS systems
  - Task automation using scripts
  - Bulk data import or export
- NSO provides a NETCONF client:
  - netconf-console: client application in Python
  - Any NETCONF client can be used



NSO Automation Integration
Cisco NSO Web Integration

- Use cases:
  - Integrate with existing EMS / NMS systems
  - Integrate with existing web-based management portals
- APIs:
  - JSON RPC for more complex integration requiring maintenance of session state
  - REST/RESTCONF for simple single transactions



REST example using curl and a GET request:

curl -u admin:admin http://localhost:8080/api/running/devices/device/PE13/config/router?deep&select=bgp;ospf

NSO Automation Integration
Cisco NSO REST API Example

```
cisco@nso:~$ curl -u admin:admin
http://localhost:8080/api/running/devices/device/PE11/config/ios:router/bgp?deep
<collection xmlns:y="http://tail-f.com/ns/rest">
  <bgp>
    <as-no>1</as-no>
    <bgp>
      <bestpath>
      </bestpath>
      <log-neighbor-changes/>
      <nexthop>
      </nexthop>
    </bgp>
    <distance>
    </distance>
    <neighbor>
      <id>10.0.0.101</id>
    </neighbor>
...
```

Use prefixes where needed

Use deep to display all child elements

HTTP methods:
- GET to retrieve resources
- PUT to replace resources
- POST to create resources
- DELETE to delete resources

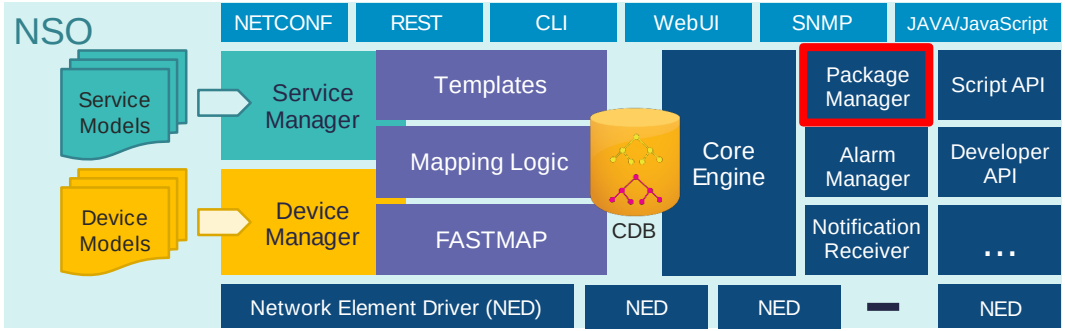
- XML encoded data received or sent by default
- JSON can be used instead of XML if desired

NSO is moving towards RESTCONF

Package Manager

Package Manager

- Loads and manages packages
- All additions to core NSO functionality are deployed as packages:
  - Simplifies deployment
  - Enables Hot-deploy/redeploy/upgrade
- Primary package types:
  - Services
  - Device NED's



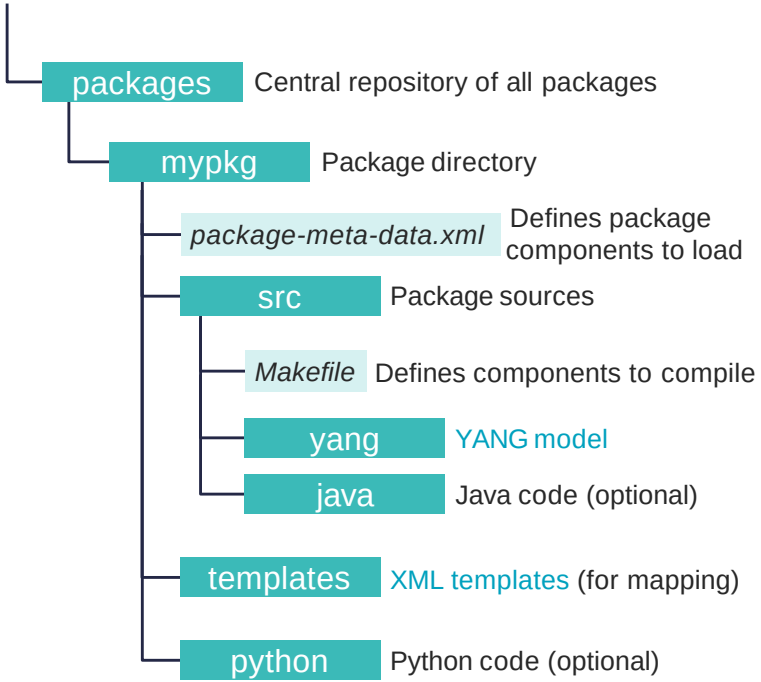
Package Structure

- Dedicated directory per package
- Package's configuration schema defined as YANG model

  `./mypkg/src/yang/mypkg.yang`
- Device configuration

  `./mypkg/templates/mypkg.xml`
- Java or Python code to augment the default functionality

  `./mypkg/src/java/src/com/acme/mypkg/mypkg.java`



Package Management

- Create package skeleton: ***ncs-make-package***
  - Simplifies initial package creation by creating a skeleton directory and file structure
- Develop package:
  - Use a text editor or an appropriate IDE to develop the package by modifying and creating the necessary files (YANG, Java, Python, XML templates)
- Compile packages: ***make***
  - Includes syntax and semantic verification of package definitions
  - Reads the ***Makefile*** to determine the components to compile
- Reload packages: ***packages reload*** (in NSO CLI)
  - Activates new packages or upgrades existing packages
  - Reads the ***package-meta-data.xml*** file to determine which components to load

Create a Package Skeleton

- Use ***ncs-make-package*** to create a development starting point from a package template (i.e. skeleton)
- Available package templates:
  - Service skeleton: template based (XML), java or python based
  - NED package
  - Data provider package
- Options:
  - Specify a destination directory
  - Etc.

Compile Packages

```
root@nso:/var/opt/ncs/packages$ cd cisco-ios/src
root@nso:/var/opt/ncs/packages/cisco-ios/src$ make
cd java && ant -q all

BUILD SUCCESSFUL
Total time: 0 seconds
cd ../netsim && make all
make[1]: Entering directory
`/var/opt/ncs/packages/cisco-ios/netsim'  make[1]:
Nothing to be done for `all'.
make[1]: Leaving directory `/var/opt/ncs/packages/cisco-ios/netsim'

root@nso:/var/opt/ncs/packages/cisco-ios/src$
```

- Compile packages to ensure they are compiled by the correct version of the compiler
- You may also do make clean and then make to perform a complete recompile of the package

Reload Packages

```
admin@ncs# packages reload
>>> System upgrade is starting.
>>> Sessions in configure mode must exit to operational mode.
>>> No configuration changes can be performed until upgrade has completed.
reload-result {
    package cisco-ios
    result true
}
reload-result {
    package cisco-iosxr
    result true
}

admin@ncs#
```

- Depending on the number of packages it may take some time for the package upgrade process to complete
- It may take even longer in case a package upgrade is performed in a live system with multiple service packages with many service instances

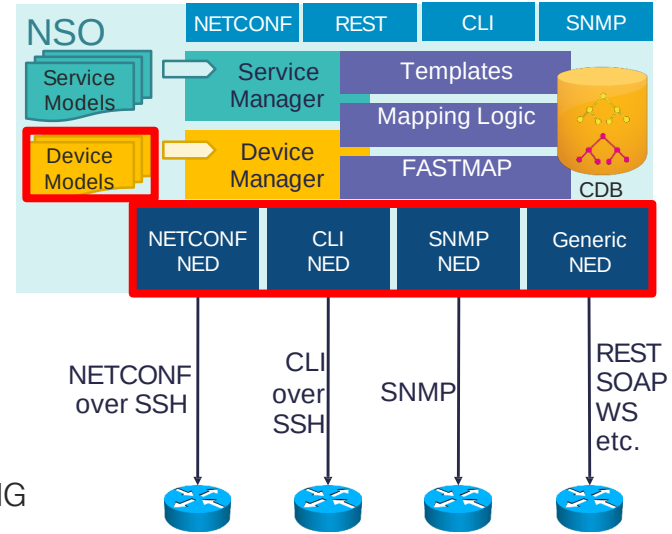
Verifying Packages

```
admin@ncs# show packages package package-version
             PACKAGE
NAME         VERSION
----------------------
cisco-ios    4.0.4
cisco-iosxr  4.1

admin@ncs#
```

- Check if all the required packages are loaded using the show packages command

Network Element Drivers

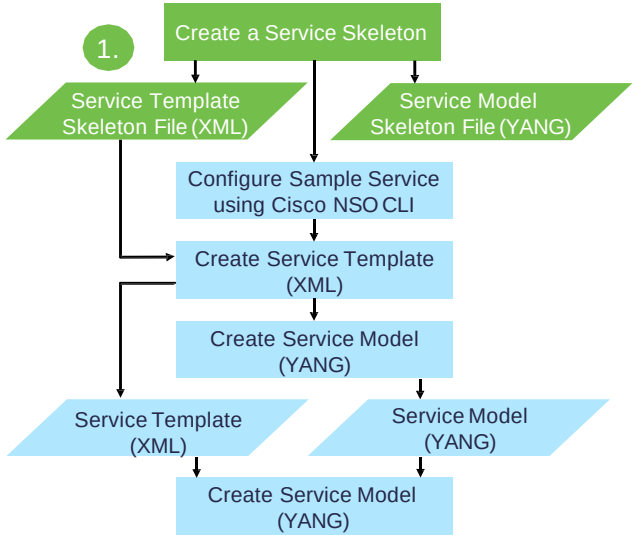
- Used to communicate southbound
- Model driven:
  - YANG device models
- Deployed as NSO packages
- NED types:
  - NETCONF NED: YANG model
  - CLI NED: YANG model
    + YANG extensions to map CLI and YANG
    + Java code for SSH communication
  - SNMP NED: YANG model
  - Generic NED: YANG model
    + framework for an arbitrary NED implementation



Service Package

Creating a Service Package

1. Create a package skeleton
2. Use the Cisco NSO CLI to configure a sample service
3. Create the service template
4. Create the service model in YANG
5. Compile and deploy the package

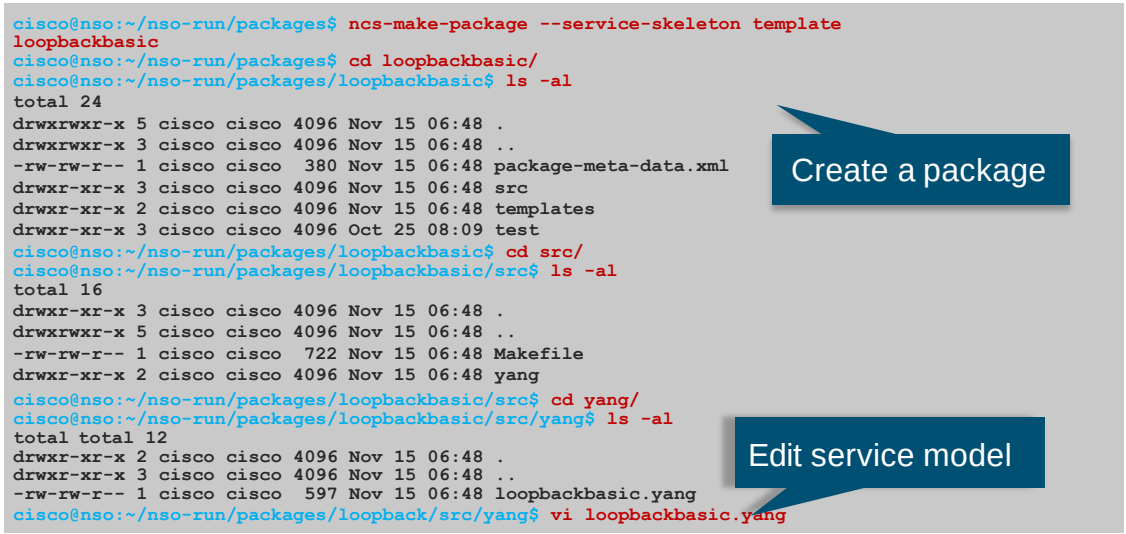


Step 1: Create a Package Skeleton

```
cisco@nso:~/nso-run$ cd packages/
cisco@nso:~/nso-run/packages$ ncs-make-package --help
Usage: ncs-make-package [options] package-name

    ncs-make-package --netconf-ned DIR package-name
    ncs-make-package --snmp-ned DIR package-name
    ncs-make-package --data-provider-skeleton package-name
    ncs-make-package --generic-ned-skeleton package-name
    ncs-make-package --erlang-skeleton package-name
    ncs-make-package --service-skeleton TYPE package-name

        where TYPE is one of:
            java                Java based service
            java-and-template   Java service with template
            python              Python based service
            python-and-template Python service with template
            template            Template service (no code)

    ADDITIONAL OPTIONS
    --dest DIR
    --build
    --verbose
    -h | --help
```

- Service Skeleton Types:
  - Customize service skeleton depending on requirements (only template, python code…)

Step 2: Check the YANG File



- Preferably use an IDE for YANG which has syntax highlighting and YANG validation capability (e.g. Eclipse, Visual Studio Code)

Step 3: Check the Template XML File


```
cisco@nso:~/nso-run/packages/loopbackbasic/src/yang$ cd ..
cisco@nso:~/nso-run/packages/loopbackbasic/src$ cd ..
cisco@nso:~/nso-run/packages/loopbackbasic$ cd templates/
cisco@nso:~/nso-run/packages/loopbackbasic/templates$ ls -al
total 12
drwxr-xr-x 2 cisco cisco 4096 Nov 15 06:48 .
drwxrwxr-x 5 cisco cisco 4096 Nov 15 06:48 ..
-rw-rw-r-- 1 cisco cisco  739 Nov 15 06:48 loopbackbasic-template.xml
cisco@nso:~/nso-run/packages/loopbackbasic/templates$ vi loopbackbasic-template.xml
```

Edit device template

- The template maps to actual device configurations
- Contains mapping for each device type used to implement the service

Things to Learn?

- How to build the device templates
- How to design a YANG service model
- Use the "top-down" or "bottom-up" approach, depending on the design environment

Top down:

1. Build the YANG service model based on service description and parameters
2. Manually configure the new service using the Cisco NSO CLI
3. Export the configuration into XML format
4. Replace exact values with variables → XML service template

Bottom up:

1. Manually configure the new service using the Cisco NSO CLI
2. Export the configuration into XML format
3. Replace exact values with variables →XML service template
4. Build the YANG service model around the created template(s)

Simple Example – Loopback Service

Cisco IOS Example

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

Cisco IOS XR Example

```
interface Loopback0
 ipv4 address 2.2.2.2 255.255.255.255
!
```

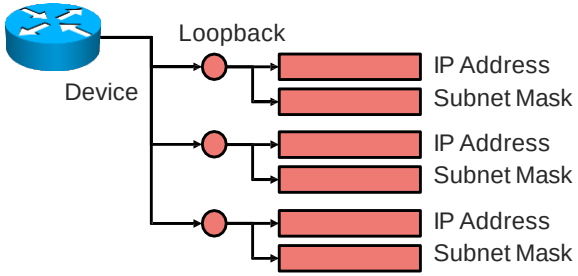
Juniper Junos Example

```
lo0 {
    unit 0 {
        family inet {
            address 3.3.3.3/32;
        }
    }
}
```

- Service requirements:
  - Configure loopback interfaces:
    - IP address
    - Subnet mask
  - Platforms: Cisco IOS, Cisco IOS XR, Juniper Junos

- Note: This example is used throughout the remainder of this lesson to illustrate how to create a simple template-based service model.

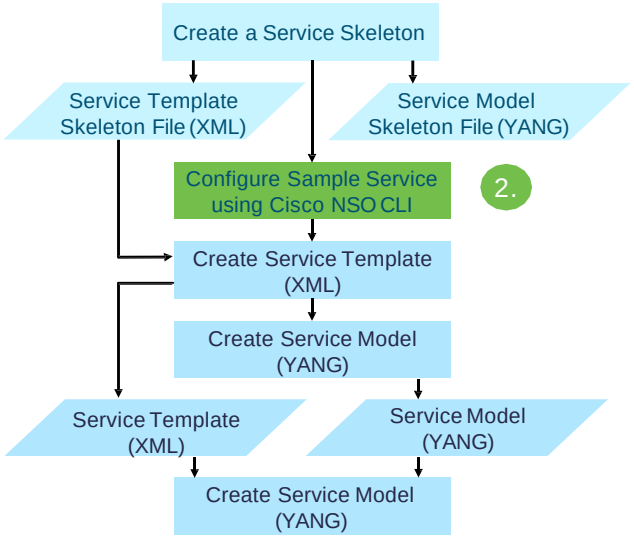
Informal Service Model Defining Service Parameters




```
/device
/device/loopback-number
/device/loopback-number/ip-address
/device/loopback-number/mask
```

Creating a Service Package

1. Create a package skeleton
2. Use the Cisco NSO CLI to configure a sample service
3. Create the service template
4. Create the service model in YANG
5. Compile and deploy the package



Example: Configure a Loopback Interface

```
admin@ncs(config)# devices device ios0 config ios:interface Loopback 0 ip address 1.1.1.1 255.255.255.255
admin@ncs(config)# top
admin@ncs(config)# devices device iosxr0 config cisco-ios-xr:interface Loopback 0 ipv4 address 2.2.2.2 255.255.255.255
admin@ncs(config)# top
admin@ncs(config)# devices device junos0 config junos:configuration interfaces interface lo0 unit 0 family inet address 3.3.3.3/32
```

- Notice that the NEDs for Cisco IOS, Cisco IOS XR, and Juniper Junos expose slightly different syntax due to different device CLIs
- Central point of management prepends another layer to the configuration hierarchy: devices
- Repeat the service configuration step for each type of device in your environment

Example: Check Candidate Configuration

- Display candidate configuration changes
- “+” indicates configuration items that will be written to CDB
- “–” indicates configuration items that will be deleted from CDB

```
admin@ncs% commit dry-run
cli {
    local-node {
        data  devices {
                  device ios0 {
                      config {
                          ios:interface {
                              Loopback 0 {
                                  ip {
                                      address {
                                          primary {
            +                                 address 1.1.1.1;
            +                                 mask 255.255.255.255;
                                          }
                                      }
                                  }
                              }
                          }
                      }
                  device iosxr0 {
                      config {
                          cisco-ios-xr:interface {
                              Loopback 0 {
                                  ipv4 {
                                      address {
            +                             ip 2.2.2.2;
            +                             mask 255.255.255.255;
                                      }
                                  }
                              }
                          }
                      }
                  device junos0 {
                      config {
                          junos:configuration {
                              interfaces {
                                  interface lo0 {
                                      unit 0 {
                                          family {
                                              inet {
            +                                     address 3.3.3.3/32;
                                              }
                                          }
                                      }
                                  }
                              }
                          }
                      }
              }
          }
      }
```

Example: Check Native Configuration

- Up-to three configuration verification stages:
  - Cisco NSO validation
  - Device validation before configuration activation
  - Device validation at configuration activation time
- Any one of the verification stages failing will result in all changes being reverted on all devices that were part of the same transaction

Example: Check Native Configuration Cont.


```
admin@ncs% commit dry-run outformat native
native {
    device {
        name ios0
        data interface Loopback0
              ip address 1.1.1.1 255.255.255.255
             exit
    }
    device {
        name iosxr0
        data interface Loopback 0
              ipv4 address 2.2.2.2
255.255.255.255
             exit
    }
```

```
    device {
        name junos0
        data <rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
                  message-id="1">
               <edit-config xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
                 <target>
                   <candidate/>
                 </target>
                 <test-option>test-then-set</test-option>
                 <error-option>rollback-on-error</error-option>
                 <config>
                   <configuration xmlns="http://xml.juniper.net/xnm/1.1/xnm">
                     <interfaces>
                       <interface>
                         <name>lo0</name>
                         <unit>
                           <name>0</name>
                           <family>
                             <inet>
                               <address>
                                 <name>3.3.3.3/32</name>
                               </address>
                             </inet>
                           </family>
                         </unit>
                       </interface>
                     </interfaces>
                   </configuration>
                 </config>
               </edit-config>
             </rpc>
    }
}
```

Example: Check XML Configuration

- Display changes in XML format as a starting point for the service template

```
admin@ncs% commit dry-run outformat xml
result-xml {
    local-node {
        data <devices xmlns="http://tail-f.com/ns/ncs">
               <device>
                 <name>ios0</name>
                 <config>
                   <interface xmlns="urn:ios">
                     <Loopback>
                       <name>0</name>
                       <ip>
                         <address>
                           <primary>
                             <address>1.1.1.1</address>
                             <mask>255.255.255.255</mask>
                           </primary>
                         </address>
                       </ip>
                     </Loopback>
                   </interface>
                 </config>
               </device>
               <device>
                 <name>iosxr0</name>
                 <config>
                   <interface xmlns="http://tail-f.com/ned/cisco-ios-xr">
                     <Loopback>
                       <id>0</id>
                       <ipv4>
                         <address>
                           <ip>2.2.2.2</ip>
                           <mask>255.255.255.255</mask>
                         </address>
                       </ipv4>
                     </Loopback>
                   </interface>
                 </config>
               </device>
```

Device Types → Different NEDs → Different XML namespaces

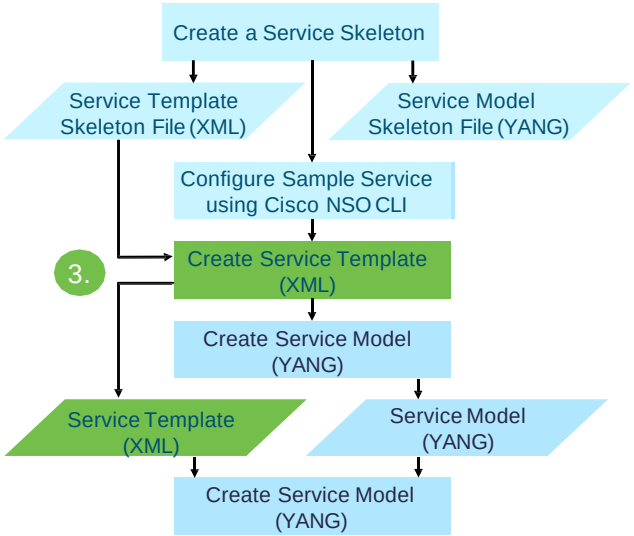
```
        <device>
              <name>junos0</name>
              <config>
                <configuration xmlns="http://xml.juniper.net/xnm/1.1/xnm">
                  <interfaces>
                    <interface>
                      <name>lo0</name>
                      <unit>
                        <name>0</name>
                        <family>
                          <inet>
                            <address>
                              <name>3.3.3.3/32</name>
                            </address>
                          </inet>
                        </family>
                      </unit>
                    </interface>
                  </interfaces>
                </configuration>
              </config>
            </device>
          </devices>
    }
}
```

- If looking at existing configuration after it has already been committed:

show devices device iosxr0 config cisco-ios-xr:interface Loopback | display xml

Creating a Service Package

1. Create a package skeleton
2. Use the Cisco NSO CLI to configure a sample service
3. Create the service template
4. Create the service model in YANG
5. Compile and deploy the package



Creating the Service Template

```
cisco@nso:~/ncs-run$ cd packages/
cisco@nso:~/ncs-run/packages$ cd loopbackbasic
cisco@nso:~/ncs-run/packages/loopbackbasic$ cd templates
cisco@nso:~/ncs-run/packages/loopbackbasic/templates$ ls -l
total 4
-rw-rw-r-- 1 cisco cisco 739 Nov 15 06:48 loopbackbasic-
template.xml
cisco@nso:~/ncs-run/packages/loopbackbasic/templates$ more  loopbackbasic-
template.xml
<config-template xmlns="http://tail-f.com/ns/config/1.0"
                 servicepoint="loopbackbasic">
  <devices xmlns="http://tail-f.com/ns/ncs">
    <device>
      <!--
          Select the devices from some data structure in the service
          model. In this skeleton the devices are specified in a leaf-list.
          Select all devices in that leaf-list:
      -->
      <name>{/device}</name>
      <config>
        <!--
            Add device-specific parameters here.
            In this skeleton the service has a leaf "dummy"; use that
            to set something on the device e.g.:
            <ip-address-on-device>{/dummy}</ip-address-on-device>
        -->
      </config>
    </device>
  </devices>
</config-template>
cisco@nso:~/ncs-run/packages/loopback/templates$ vi  loopbackbasic-template.xml
```

1. Edit the skeleton XML service template file
2. Define variables in curly brackets: {variable}
3. Variables are mapped to the service model:
4. Use the path based on the informal service model hierarchy

Example: Convert XML Configuration to Template

- IOS:

```
admin@ncs% commit dry-run outformat xml
result-xml {
    local-node {
        data <devices xmlns="http://tail-f.com/ns/ncs">
               <device>
                 <name>ios0</name>
                 <config>
                   <interface xmlns="urn:ios">
                     <Loopback>
                       <name>0</name>
                       <ip>
                         <address>
                           <primary>
                             <address>1.1.1.1</address>
                             <mask>255.255.255.255</mask>
                           </primary>
                         </address>
                       </ip>
                     </Loopback>
                   </interface>
                 </config>
               </device>
```

Variable

Variable

Fixed

```
<config-template xmlns="http://tail-f.com/ns/config/1.0"
                 servicepoint="loopbackbasic">
  <devices xmlns="http://tail-f.com/ns/ncs">
    <device>
      <name>{/device}</name>
      <config>

        <interface xmlns="urn:ios">
          <Loopback>
            <name>{/loopback-number}</name>
            <ip>
              <address>
                <primary>
                  <address>{/ip-address}</address>
                  <mask>255.255.255.255</mask>
                </primary>
              </address>
            </ip>
          </Loopback>
        </interface>
```

Example: Convert XML Configuration to Template

- IOS XR:

```
<device>
   <name>iosxr0</name>
   <config>
     <interface xmlns="http://tail-f.com/ned/cisco-ios-xr">
       <Loopback>
         <id>0</id>
         <ipv4>
           <address>
             <ip>2.2.2.2</ip>
             <mask>255.255.255.255</mask>
           </address>
         </ipv4>
       </Loopback>
     </interface>
   </config>
 </device>
```

Variable

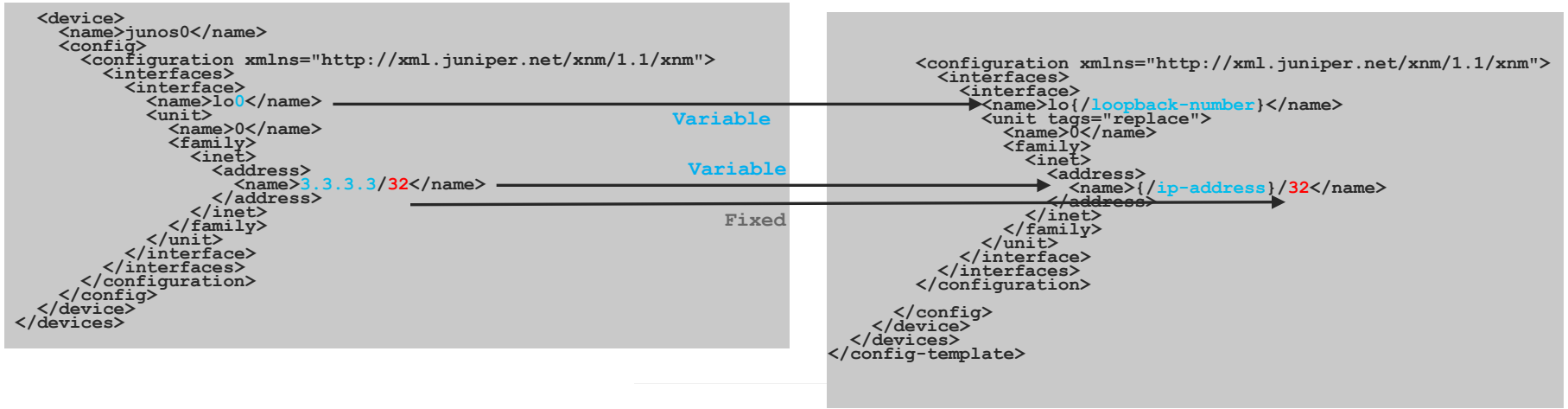
Variable

Fixed

```
<interface xmlns="http://tail-f.com/ned/cisco-ios-xr">
  <Loopback>
    <id>{/loopback-number}</id>
    <ipv4>
      <address>
        <ip>{/ip-address}</ip>
        <mask>255.255.255.255</mask>
      </address>
    </ipv4>
  </Loopback>
</interface>
```

Example: Convert XML Configuration to Template

- JUNOS:




```
<device>
  <name>junos0</name>
  <config>
    <configuration xmlns="http://xml.juniper.net/xnm/1.1/xnm">
      <interfaces>
        <interface>
          <name>lo0</name>
          <unit>
            <name>0</name>
            <family>
              <inet>
                <address>
                  <name>3.3.3.3/32</name>
                </address>
              </inet>
            </family>
          </unit>
        </interface>
      </interfaces>
    </configuration>
  </config>
</device>
</devices>
```

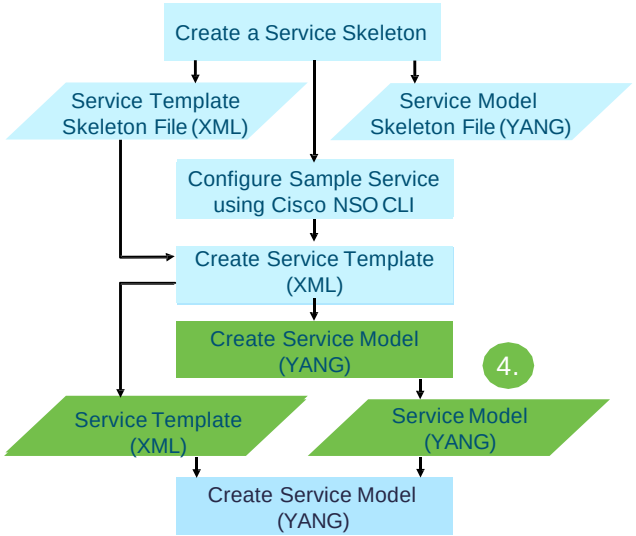
```
<configuration xmlns="http://xml.juniper.net/xnm/1.1/xnm">
  <interfaces>
    <interface>
      <name>lo{/loopback-number}</name>
      <unit tags="replace">
        <name>0</name>
        <family>
          <inet>
            <address>
              <name>{/ip-address}/32</name>
            </address>
          </inet>
        </family>
      </unit>
    </interface>
  </interfaces>
</configuration>

</config>
</device>
</devices>
</config-template>
```

Template Options (Cont.)

- The template usage can be modified if desired:
  - default XPath context: by default XPath root is set to the root of service instance data – it can be changed to another context to simplify work
  - conditional statements: Sometimes it is necessary to control which parts of a template that should be evaluated. The when attribute makes it possible to set a conditional statement that controls if the sub-tree should be evaluated or not
  - loop statements: Sometimes statements in a sub-tree needs to be applied several times. The attribute for each can be used to accomplish this iteration

Creating a Service Package

1. Create a package skeleton
2. Use the Cisco NSO CLI to configure a sample service
3. Create the service template
4. Create the service model in YANG
5. Compile and deploy the package



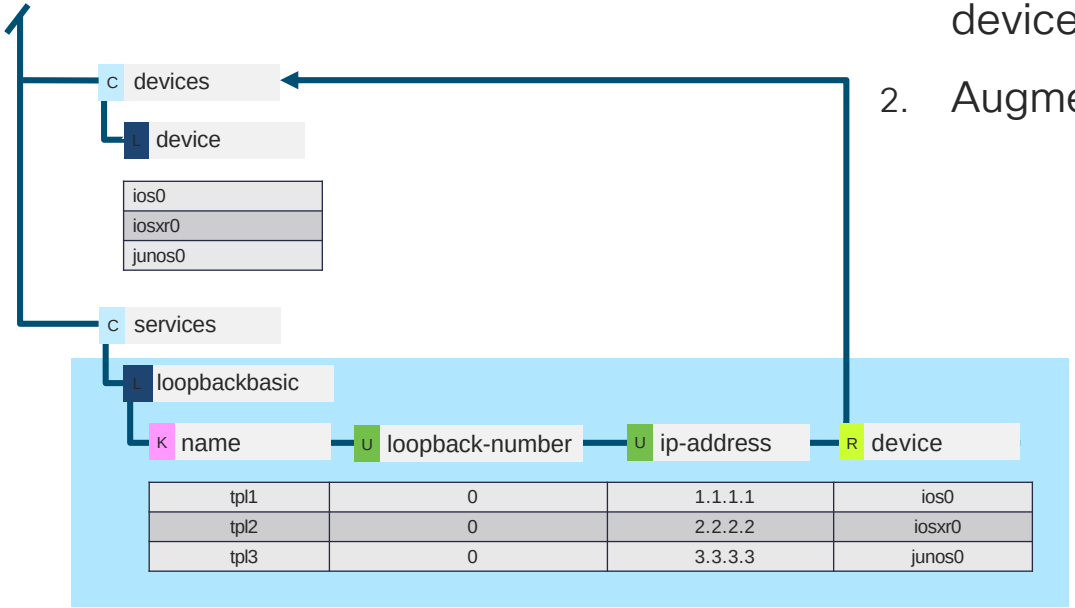
Step 1: Creating the Service Model:
Start with Skeleton Model

```
namespace "http://com/example/loopbackbasic";
 prefix loopbackbasic;

import ietf-inet-types { prefix inet; }
import tailf-ncs { prefix ncs; }

augment /ncs:services {
  list loopbackbasic
    {  key name;

    uses ncs:service-data;
    ncs:servicepoint
    "loopbackbasic";

    leaf name {
      type string;
    }

    // may replace this with other ways of refering to the devices.
    leaf-list device {
      type leafref {
        path "/ncs:devices/ncs:device/ncs:name";
      }
    }

    // replace with your own stuff here
    leaf dummy {
      type inet:ipv4-address;
     }
   }
  }
}
```

Replace this section with your service model.

1. Change directory to ../src/yang
2. Edit the skeleton service model
3. The module augments the Cisco NSO services module
4. Devices are already references
5. Replace the dummy leaf with your service model

Step 2: Creating the Service Model:
Informal Service Model



1. Reuse existing resources (e.g. devices)
2. Augment the services module

Step 2: Creating the Service Model:
Top Level List

L loopbackbasic

K name	U loopback-number	U ip-address	R device
tpl1	0	1.1.1.1	ios0
tpl2	0	2.2.2.2	iosxr0
tpl3	0	3.3.3.3	junos0

```
module loopback {
  namespace "http://com/example/loopback";

  prefix loopback;

  import ietf-inet-types { prefix inet; }
  import tailf-ncs { prefix ncs; }
  import tailf-common { prefix tailf; }

  augment /ncs:services {
    list loopbackbasic {
      leaf name { ... }
      leaf device { ... }
      leaf loopback-number { ... }
      leaf ip-address { ... }
    }

  }
}
```

- Add your data model to the skeleton based on the informal design you have created earlier:
- Leaf name is used as **key** for the list of services
- Leafs device, loopback-number and ip-address are leafs identifying the loopback interface parameters

Step 3: Creating the Service Model:
Service Instance ID

L loopbackbasic

K name	U loopback-number	U ip-address	R device
tpl1	0	1.1.1.1	ios0
tpl2	0	2.2.2.2	iosxr0
tpl3	0	3.3.3.3	junos0

```
...
    list loopbackbasic {
      key name;

      uses ncs:service-data;
      ncs:servicepoint
      "loopbackbasic";

      leaf name {
        tailf:info "Service Instance Name";
        type string;
      }
...
```

- Tell NSO that this is a service
- The custom name for the service is used as unique identifier in the list
- Optionally use the NSO-specific annotations to provide textual description used in NSO CLI and WebUI
- One implicit requirement: name is mandatory as it is the key

Step 3: Creating the Service Model:
Implement Service Model

L loopbackbasic

K name	U loopback-number	U ip-address	R device
tpl1	0	1.1.1.1	ios0
tpl2	0	2.2.2.2	iosxr0
tpl3	0	3.3.3.3	junos0

```
...
  list loopbackbasic {
...
      leaf device {
        tailf:info "Router name";
        mandatory true;
        type leafref {
          path "/ncs:devices/ncs:device/ncs:name";
        }
      }
...
```

- Keep the default reference to the live list of devices
- One requirement: device is mandatory

Step 3: Creating the Service Model:
Implement Service Model (Cont.)

L loopbackbasic

K name	U loopback-number	U ip-address	R device
tpl1	0	1.1.1.1	ios0
tpl2	0	2.2.2.2	iosxr0
tpl3	0	3.3.3.3	junos0

```
...
    list loopbackbasic {
...
      leaf loopback-number {
        tailf:info "Loopback Interface Number";
        mandatory true;
        type uint32;
      }
...
```

- One requirement: loopback interface number is mandatory

Step 3: Creating the Service Model:
Implement Service Model (Cont.)

L loopbackbasic

K name	U loopback-number	U ip-address	R device
tpl1	0	1.1.1.1	ios0
tpl2	0	2.2.2.2	iosxr0
tpl3	0	3.3.3.3	junos0

```
...
list loopbackbasic {
...
  leaf ip-address {
    tailf:info "Valid IP";
    mandatory true;
    type inet:ipv4-address;
  }
...
```

- The inet:ipv4-address data type limits the value to a properly formatted IPv4 address
- One requirement: ip-address is mandatory

Regular Expressions – Quick Overview


```
Matching a single character
.             dot matches any single character
[a-zA-Z0-9]   matches a single character from a range of characters

Quantifiers
char*         The preceeding character appears zero or more times
char+         The preceeding character appears one or more times
char?         The preceeding character appears zero or one time
char{n}       The preceeding character appears exactly n times
char{n,}      The preceeding character appears at least n times
char{n,m}     The preceeding character appears at least n and no more than m times

Grouping
(abc)*        The preceeding group of characters appears zero or more times

Logical operators
(abc)|(cde)   Matches at least one of the expressions on left or right of the OR operator
[^a-zA-Z0-9]  Matches a single character NOT from the specified range of characters

Escape character to match any character that has a special meaning in regular expressions: .?+*(){}[]\|
\.            Matches the dot character
\\            Matches the backslash

Multiple character escape sequences: \w \W \s \S \d \D
\d            Matches any decimal number; equals [0-9]+
\w            Matches a contiguous sequence of alphanumeric characters; equals [a-zA-Z]+
```

Regular Expressions - Example

- Goal: match valid IP addresses in the range from 10.100.0.0 to 10.199.255.255

valid	not in range	not valid IP	not valid IP
10.100.1.1	10.200.1.1	1001.100.1.1	10.100.1.500

✓ = match

✗ = no match

- Which of the above addresses will be matched by the following regular expressions

`10.\d+.\d+.\d+`	10.100.1.1 ✓	10.200.1.1 ✓	1001.100.1.1 ✓	10.100.1.500 ✓
`10\.\d+\.\d+\.\d+`	10.100.1.1 ✓	10.200.1.1 ✓	1001.100.1.1 ✗	10.100.1.500 ✓
`10\.1[0-9][0-9]\.[0-9]+\.[0-9]+`	10.100.1.1 ✓	10.200.1.1 ✗	1001.100.1.1 ✗	10.100.1.500 ✓

- Do not reinvent the wheel - reuse ietf-yang-types and ietf-inet-types modules where a number of useful data types are already defined

`(([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])`

ipv4-address data type

and

`10\.1[0-9][0-9]\.[0-9]+\.[0-9]+`	10.100.1.1 ✓	10.200.1.1 ✗	1001.100.1.1 ✗	10.100.1.500 ✗

Step 4: Creating the Service Model:
Final Service Model

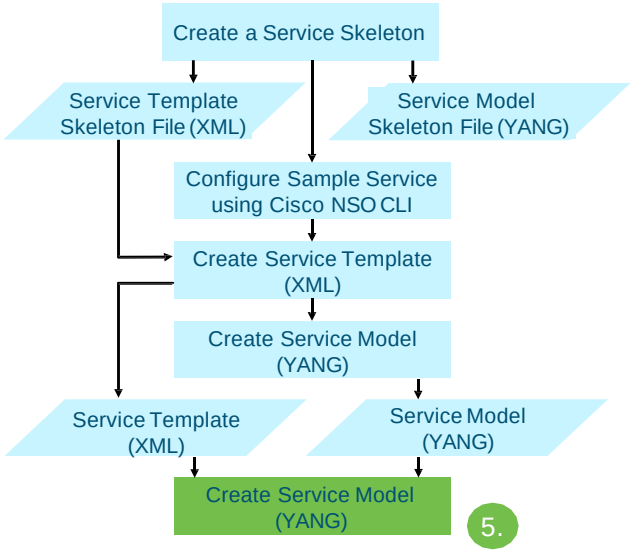
```
module loopbackbasic {
 namespace
 "http://com/example/loopbackbasic";

 prefix loopbackbasic;

 import ietf-inet-types { prefix inet; }
 import tailf-ncs { prefix ncs; }
 import tailf-common { prefix tailf; }

 augment /ncs:services {

   list loopbackbasic {

     key name;

     uses ncs:service-data;
     ncs:servicepoint
     "loopbackbasic";

     leaf name {
       tailf:info "Service Instance Name"
       mandatory true;
       type string;
     }
...
```

```
     leaf device {
       tailf:info
       "Router name";
       mandatory true;
       type leafref {
         path "/ncs:devices/ncs:device/ncs:name";
       }
     }
     leaf loopback-number {
       tailf:info "Loopback Interface Number";
       mandatory true;
       type uint32;
     }

     leaf ip-address {
      tailf:info "Valid IP";
      mandatory true;
      type inet:ipv4-address;
     }
   }
 }
```

- Parameter hierarchy, data types, restrictions, NSO-specific annotations (e.g. descriptions used by the Cisco NSO GUI and CLI)

Creating a Service Package

1. Create a package skeleton
2. Use the Cisco NSO CLI to configure a sample service
3. Create the service template
4. Create the service model in YANG
5. Compile and deploy the package



Compile and Deploy the Service Model

```
cisco@nso:~/ncs-run/packages/loopbackbasic/src/yang$ cd ..
cisco@nso:~/ncs-run/packages/loopbackbasic/src$ make
mkdir -p ../load-dir
/Users/rereich/NCS_Builds/./nso-4.5.2/bin/ncsc  `ls loopbackbasic-ann.yang  > /dev/null 2>&1 && echo "-a loopbackbasic-ann.yang"` \
-c -o ../load-dir/loopbackbasic.fxs yang/loopbackbasic.yang
cisco@nso:~/ncs-run/packages/loopbackbasic/src$
```

```
admin@ncs% request packages reload

>>> System upgrade is starting.
>>> Sessions in configure mode must exit to operational mode.
>>> No configuration changes can be performed until upgrade has completed.
>>> System upgrade has completed successfully.
```

- Compile the changes on the server
- Request the reload of packages in the Cisco NSO CLI

Parameter Restrictions Enforced Already in GUI and CLI

YANG data types and constraints are validated

```
admin@nso(config)# services loopbackbasic test device ios0 loopback-number 1 ip-address 260.
-----------------------------------------------------------------------------------^
syntax error: "260." is not a valid value.

admin@nso(config)# services loopbackbasic test device ios0 loopback-number 1 ip-address 1.1.1.1
admin@ncs(config-loopbackbasic-test)# top
admin@nso(config)# commit
Commit complete.
admin@nso(config)# exit
admin@nso# show running-config services loopbackbasic
test
services loopbackbasic test
 device          ios0
 loopback-number 1
 ip-address      1.1.1.1
!
```

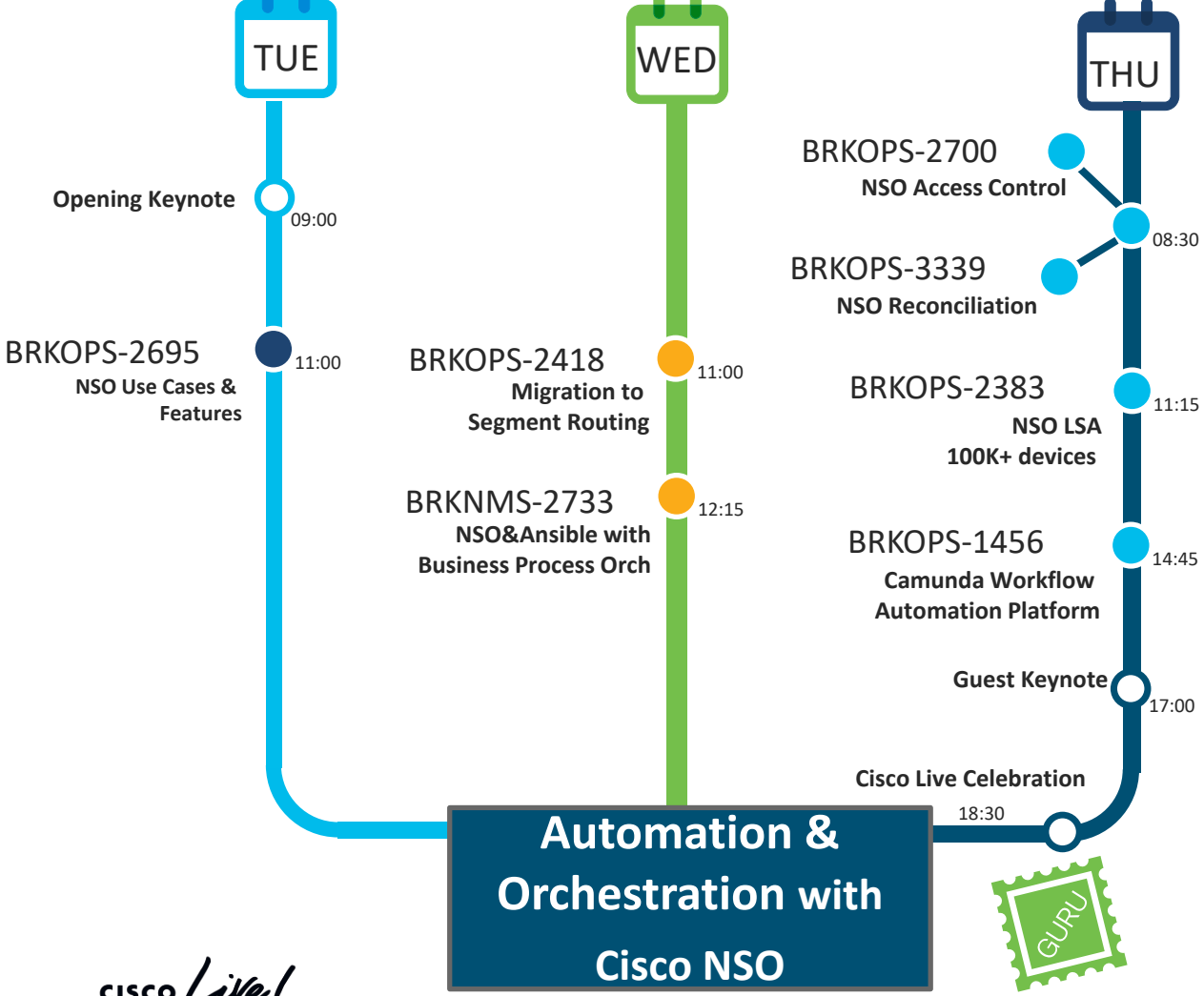
- First configuration validation stage

LAB Exercise 2

Q & A

Summary

- Network Automation Challenges
- NSO Architecture
- Device Manager
- YANG data modeling
- NSO Service Manager
- NSO Package Manager
- Design a service with NSO



OPS

Operations Track

Continue Your Education

- Related sessions
  - The 3 Stages of Network Automation and Orchestration - PSOSPG-2941
  - Automated NSO Delivery with CI/CD - LTRNMS-2984
- DevNet
  - Quick Wins of using NSO - DEVLIT-4019
  - Get Started Automating Your Network with Cisco NSO - DEVNET-4805
- Lab
  - Introduction to Cisco Network Service Orchestrator (NSO) - LABNMS-2011
  - Real-time Services Automation with NSO and Model-Driven Telemetry - LABOPS-2320

Call to Action – We want you to...

- Check out DevNet's Learning Labs on NSO https://learninglabs.cisco.com/
- DevNet NSO page (download for free) https://developer.cisco.com/site/nso/
- Become NSO Developer Hub member : https://communities.cisco.com/community/developer/nso-developer-hub
- Have NSO always with you: Play with a local-install on your laptop

The bridge between Network Automation and Business

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.

cisco Live!

Continue your education

Demos in the Cisco Showcase

Walk-In Labs

Meet the Engineer 1:1 meetings

Related sessions

Thank you

cisco Live!


cisco
You make possible

Backup Slides

Network Automation Challenges (extra)

NSO Architecture (extra)

Automatic Service Package Upgrade

- Every package reload action triggers a service upgrade process
- Automatic service upgrade succeeds in the following cases:
  - Deleted nodes: always succeeds; deletes nodes from data tree
  - Added nodes: succeeds if not mandatory without default
  - Re-ordered nodes: always succeeds
  - Type changes: succeeds if smart type conversion succeeds or there is a default value
  - Key changes: succeeds if modified key leaf is still unique
  - Default values: always succeeds and changes values only to those leafs that had a default value

Device Configuration Management (extra)

Cloning a Device

- Devices can be instantiated from other devices
- These devices can be immediately configured
- All configuration succeeds within NSO and nothing is sent to devices (southbound-locked state!)
- All device configuration is cloned!

```
admin@nso(config)# devices device PE12 instantiate-from-other-device device-name PE11
admin@nso(config)# devices device PE13 instantiate-from-other-device device-name PE11
admin@nso(config)# devices device PE12 address 10.1.1.12
admin@nso(config)# devices device PE13 address 10.1.1.13
admin@nso(config)# commit
Commit complete.
```

Create a Device Using Templates

- Devices can be instantiated from templates
- These devices can be immediately configured
- All configuration succeeds due to southbound-locked state
- Only template specific configuration is applied

```
admin@nso(config)# devices device www5 apply-template template-name std-web-server
admin@nso(config)# devices device www5 address 127.0.0.1 port 23456 authgroup default
admin@nso(config)# commit
```

Load Using “ncs_load“ Command

- A convenient way of loading and saving configuration
- ncs_load without any options it will print the current configuration
- Load (-l) option uploads XML contents to the CDB
- Merge (-m) option updates the existing device list
- Example of uploading and merging configurations of Netsim devices to NSO:

```
admin@nso:~$ ncs-netsim ncs-xml-init > devices.xml
admin@nso:~$ ncs_load -l -m devices.xml
```

Make sure you use the -m option to merge the XML contents with the CDB. The default action is replace which would delete the entire CDB and replace it with the XML contents!

Meta Configuration – Single Device

Highlighted parameters represent the minimum required configuration for a device

```
admin@nso(config)# devices device PE11

Possible completions:
  address                     - IP address or host name for the management interface
  authgroup                   - Authentication credentials for the device
  config                      - NCS copy of the device configuration
  connect-timeout             - Timeout in seconds for new connections
  description                 - Free form textual description
  device-type                 - Management protocol for the device
  live-status-protocol        - Additional protocols for the live-tree (read-only)
  location                    - Physical location of devices in the group
  netconf-notifications       - NETCONF notifications from the device
  port                        - Port for the management interface
  read-timeout                - Timeout in seconds used when reading data
  remote-node                 - Name of remote node which connects to device
  snmp-notification-address - Notification address if different from device address
  source                      - How the device was added to NCS
  state                       - Show states for the device
  trace                       - Enable trace for this device
  write-timeout               - Timeout in seconds used when writing data
```

Meta Configuration – All Devices

```
admin@nso(config)# devices global-settings

Possible completions:
  commit-queue                 - Enables and configures the commit-queue
  commit-retries               - Retry commits on transient errors
  connect-timeout              - Timeout in seconds for new connections
  ned-settings                 - Control which device capabilities ncs uses
  out-of-sync-commit-behaviour - Specifies the behaviour of a commit operation involving a device that
  is
                                 out of sync with ncs.
  read-timeout                 - Timeout in seconds used when reading data
  report-multiple-errors       - By default, when the ncs device manager commits data southbound and
  when
                                 there are errors, we only report the first error to the operator,
                                 this  flag makes ncs report all errors reported by managed devices
  session-pool                 - Control how sessions to related devices can be
  pooled.  ssh-keep-alive      - Controls SSH keep alive settings
  trace                        - Trace the southbound communication to devices
  trace-dir                    - The directory where trace files are stored
  write-timeout                - Timeout in seconds used when writing data
```

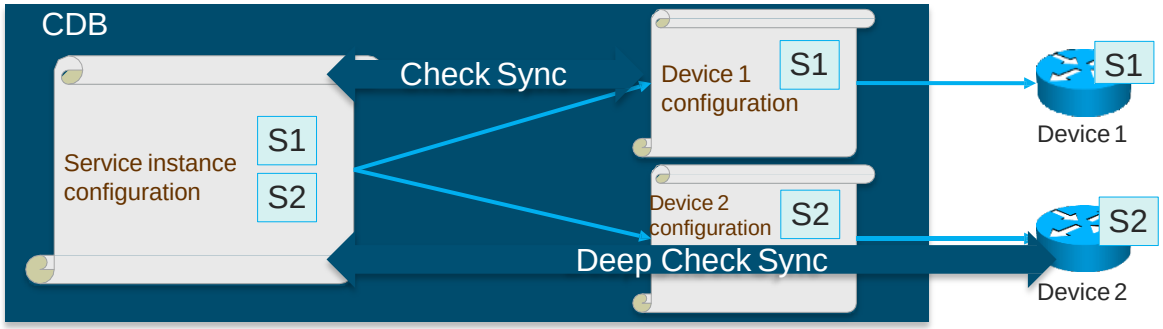
Device Action – All Devices

```
admin@nso# devices ?

Possible completions:
  check-sync           - Check if the NCS config is in sync with the device
  check-yang-modules - Check if NCS and the devices have compatible YANG modules
  clear-trace          - Clear all trace files
  commit-queues        - List of queued commits
  connect              - Set up sessions to all unlocked devices
  device               - The list of managed devices
  device-group         - Groups of devices
  disconnect           - Close all sessions to all devices
  sync                 - DEPRECATED - use sync-to or sync-from instead
  sync-from            - Synchronize the config by pulling from the devices
  sync-to              - Synchronize the config by pushing to the devices
```

Introduction to YANG (extra)

Service Manager (extra)

Service Configuration Maintenance Tools



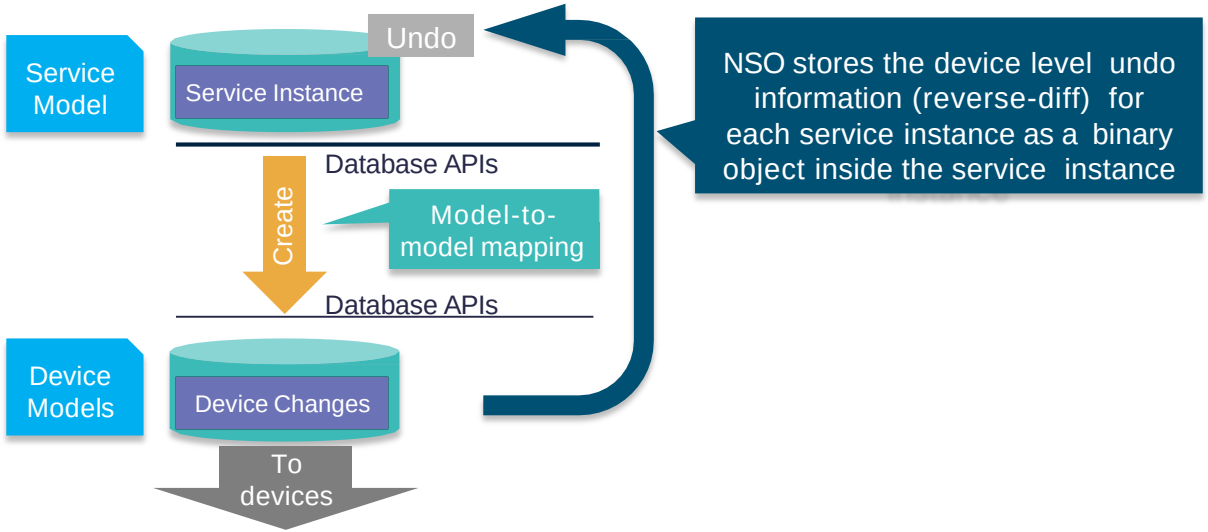

```
admin@nso# service l3mplsvpn S1 check-sync
admin@nso# service l3mplsvpn S2 deep-check-sync
```

- Internal sync check between service configuration and device-specific configuration as a result of the service-to-device mapping logic
- Extend sync check to the actual device configuration

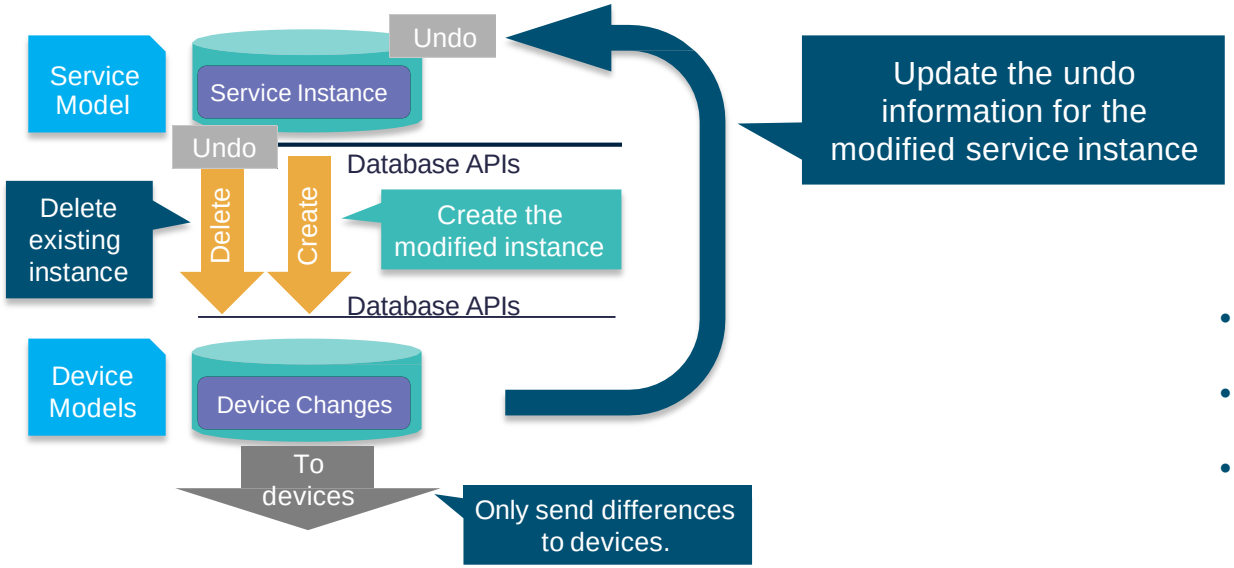
FastMap (extra)

FASTMAP: Create a Service Instance

- Create
- Modify
- Delete

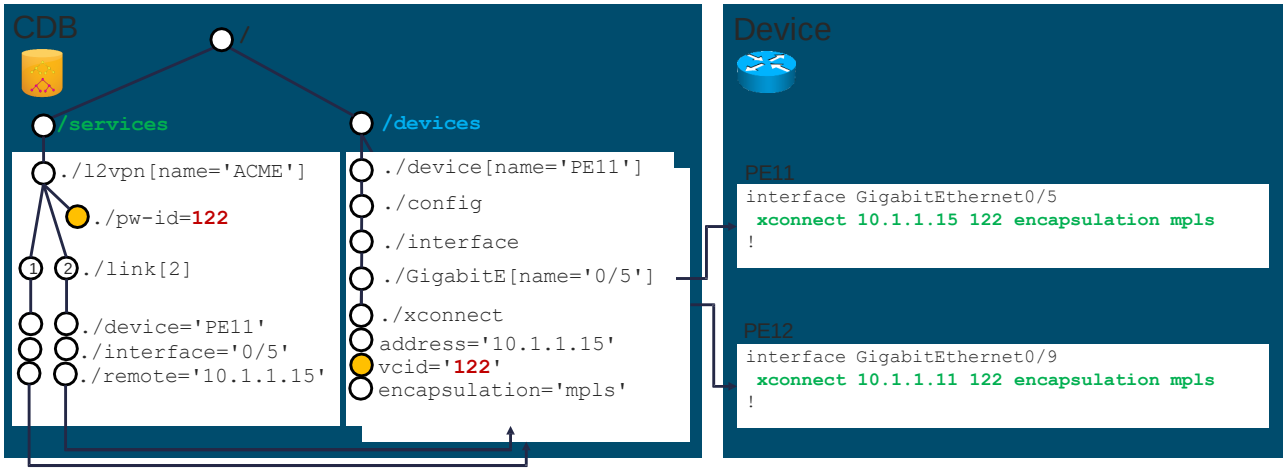


FASTMAP: Modify a Service Instance



- Create
- Modify
- Delete

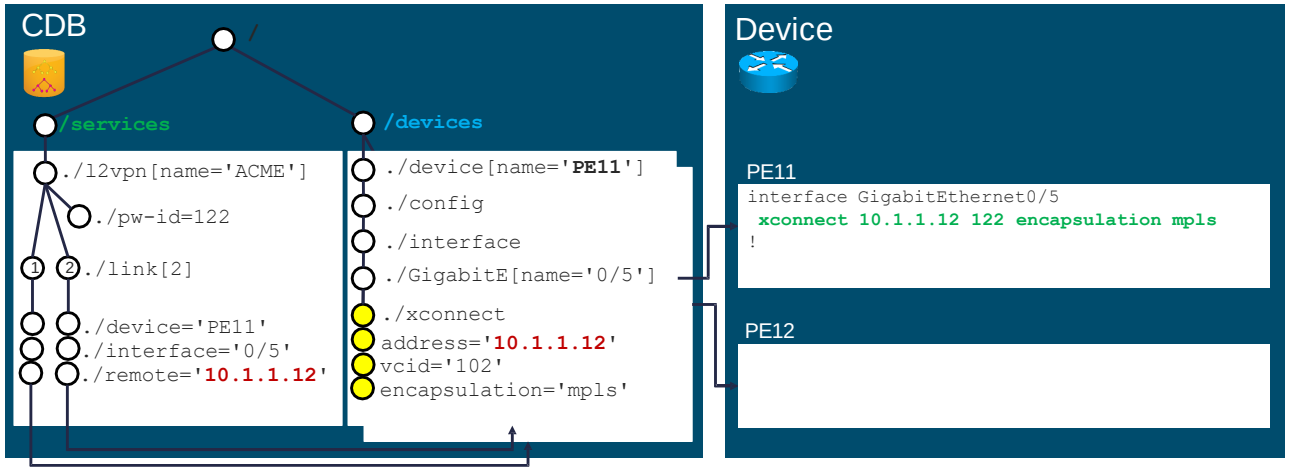
FASTMAP: Service Modify (1)



: service mapping & diff
& update reverse diff

sent to devices
CLI NED may have additional dependencies between configuration options (e.g. single line)

FASTMAP: Service Modify (2)



: service mapping & diff
& update reverse diff

Diff sent

Modify and Existing Service: Example 3

```
admin@nso# config
Entering configuration mode terminal
admin@ncs(config)# services l2vpn ACME
admin@ncs(config-l2vpn-ACME)# link Site1
admin@ncs(config-l2vpn-ACME)# intf 0/6
admin@ncs(config-link-PE12)# top
admin@ncs(config)# commit dry-run
device PE11
   config {
       ios:interface {
           GigabitEthernet 0/5 {
               xconnect {
-                  address 10.1.1.12;
-                  vcid 122;
-                  encapsulation mpls;
               }
           }
           GigabitEthernet 0/6 {
               xconnect {
+                  address 10.1.1.12;
+                  vcid 122;
+                  encapsulation mpls;
               }
           }
       }
   }
```

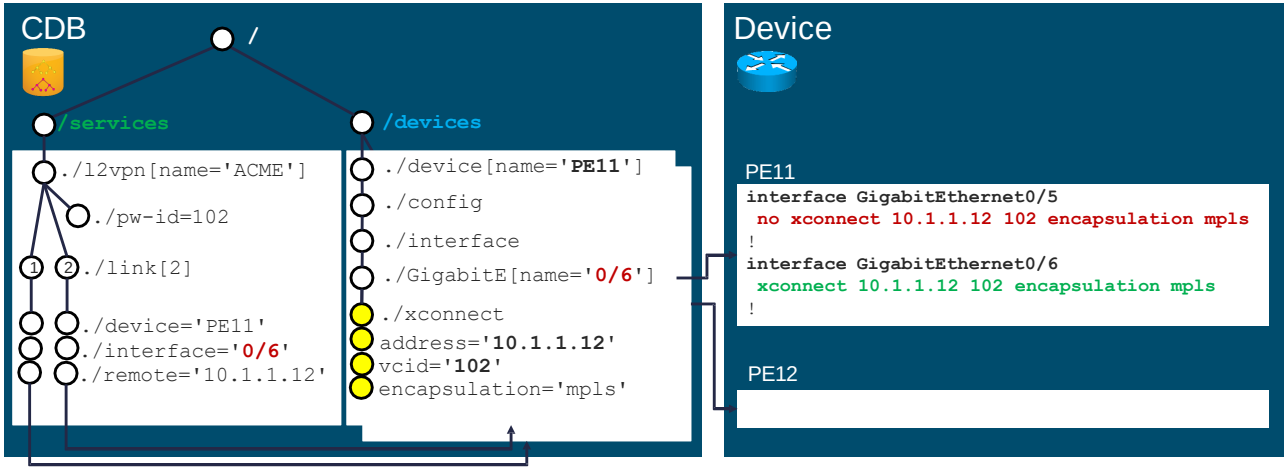
- Only one parameter changed
- CDB shows only minimal diff changes

Modify and Existing Service: Example 3 (Cont.)

```
admin@nso# config
Entering configuration mode terminal
admin@ncs(config)# services l2vpn ACME
admin@ncs(config-l2vpn-ACME)# link Site1
admin@ncs(config-l2vpn-ACME)# intf 0/6
admin@ncs(config-link-PE12)# top
admin@ncs(config)# commit dry-run outformat native
device PE11
  interface GigabitEthernet0/5
   no xconnect 10.1.1.12 122 encapsulation mpls
   exit
  interface GigabitEthernet0/6
   xconnect 10.1.1.12 122 encapsulation mpls
   exit
  exit
```

- Changed parameter affects only one device
- CLI NED may have additional dependencies between configuration options (e.g. multiple parameters in a single line)

FASTMAP: Service Modify (3)

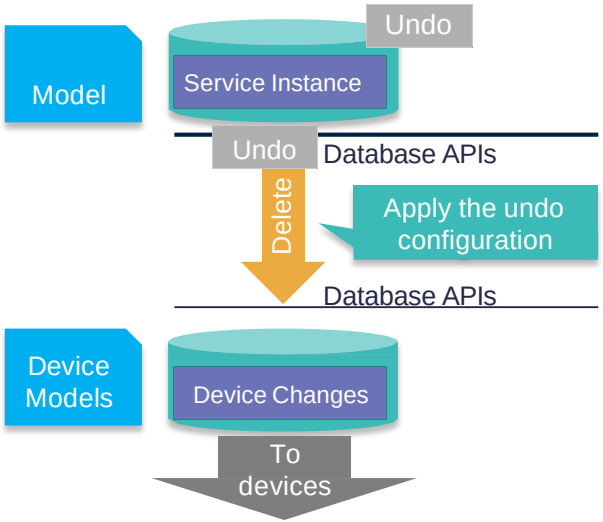


: service mapping & diff & update reverse diff

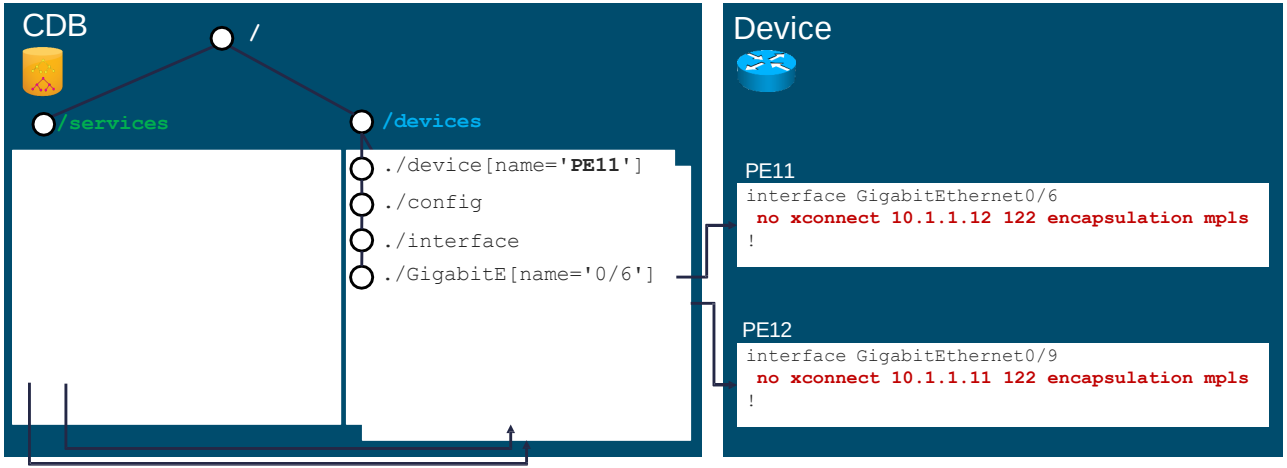
Diff of parent requires reconfiguration of all child nodes

FASTMAP: Delete a Service Instance

- Create
- Modify
- Delete



FASTMAP: Service Delete

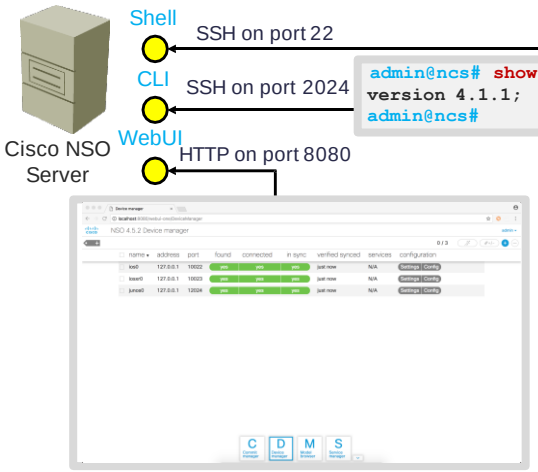


: reverse diff applied

sent to devices

Northbound API and Integration options (extra)

Cisco NSO Server User Interfaces




```
admin@ncs# show ncs-state version
version 4.1.1;
admin@ncs#
```

```
root@NCS:/opt/ncs/ncs-4.1.1/# ls -l
total 368
drwxr-xr-x  2 root root   4096 Jun 28 12:46 bin
-rw-r--r--  1 root root 140956 Jun 28 12:47 CHANGES
                ot root   4096 Jun 27 21:36 doc
                ot root   4096 Jun 28 12:46 erlang
                ot root   4096 Jun 28 12:45 etc
                ot root   4096 Jun 28 12:50 examples.ncs
drwxr-xr-x  2 root root   4096 Jun 28 12:46 include
drwxr-xr-x  3 root root   4096 Jun 28 12:45 java
drwxr-xr-x  5 root root   4096 Jun 28 12:48 lib
-rw-r--r--  1 root root 128459 Jun 28 12:47 LICENSE
drwxr-xr-x  5 root root   4096 Jun 27 21:34 man
-rw-r--r--  1 root root    541 Jul  4 14:10 ncsrc
-rw-r--r--  1 root root    509 Jul 4 14:10 ncsrc.tcsh
drwxr-xr-x  3 root root   4096 Jul 4 14:10 netsim
drwxr-xr-x 17 root root   4096 Oct 28 10:40 packages
-rwxr--r--  1 root root  17694 Jun 28 12:47 README
drwxr-xr-x  4 root root   4096 Jun 28 12:46 scripts
drwxr-xr-x  3 root root   4096 Jun 28 12:45 src
drwxr-xr-x  4 root root   4096 Jun 28 12:47 support
drwxr-xr-x  3 root root   4096 Jun 28 12:45 var
-rw-r--r--  1 root root    220 Jun 28 12:47 VERSION
```

- The *ncs.conf* file contains the server configuration with default management ports

Cisco NSO REST API Example (Cont.)

```
cisco@nso:~$ curl -u admin:admin -H "Accept: application/vnd.yang.data+json"
http://localhost:8080/api/running/devices/device/PE11/config/ios:router/bgp?deep
{
  "tailf-ned-cisco-ios:router": {
    "bgp": [
      {
        "as-no": 1,
        "address-family": {
          "ipv4": [
            {
              "af": "unicast",
              "bgp-af": {
                "bgp": {
                  "bestpath": {
                  },
                  "nexthop": {
                  }
                }
              },
              "neighbor": [
                {
                  "id": "10.1.1.13",
                  "remote-as": "1",
                  "capability": {
                  },
                  "translate-update": {
                  }
                },
...
```

Instruct NSO to reply using JSON for data encoding

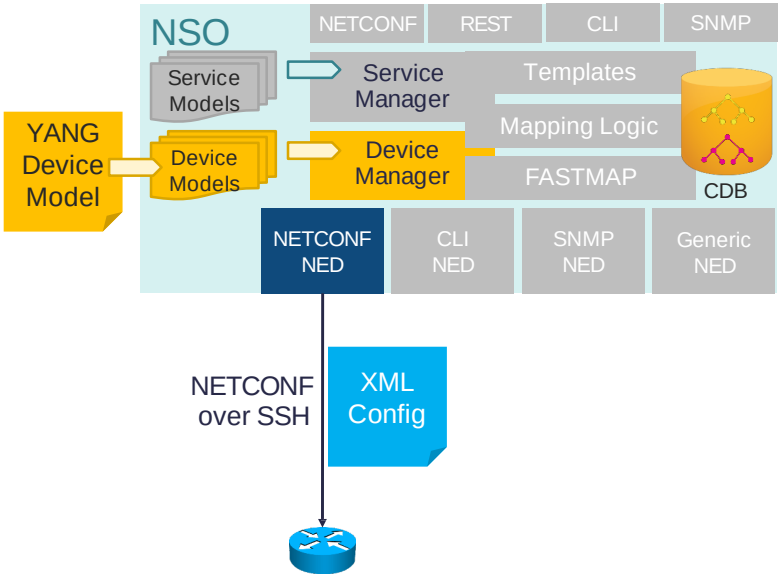
Depth options:
- Default: 5 levels
- "deep": all levels
- "shallow": 2 levels

- JSON can be used instead of XML if desired

Package Manager (extra)

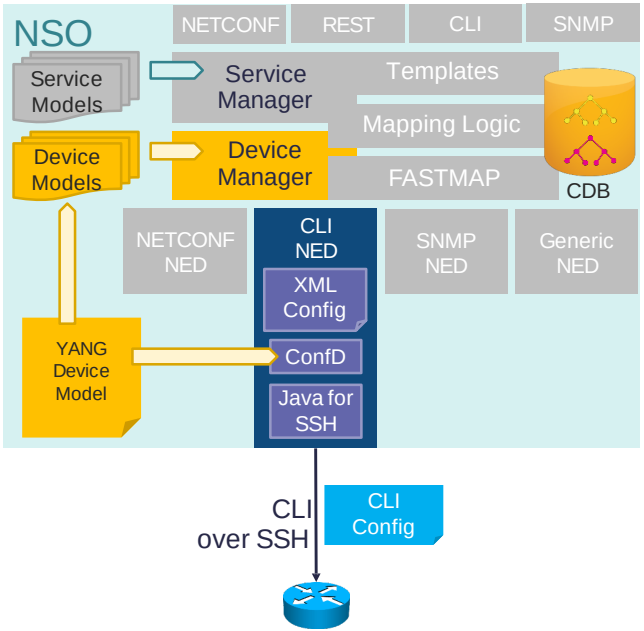
NETCONF NED

- Native NED for NSO
- No coding required
- Only YANG device model required
- Can be used with any device supporting NETCONF (e.g. by incorporating ConfD)



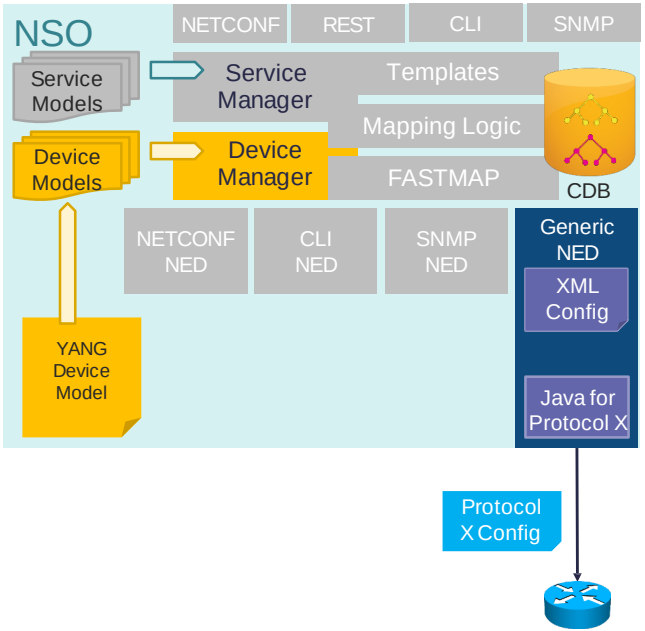
CLI NED

- YANG device model:
  - Device config options
  - CLI syntax through YANG extensions
- ConfD converts to and from between XML and CLI



Generic NED

- Custom implementation of southbound protocol
- Array of operations in the form of DOM* (Document Object Model) manipulations (JAVA) are sent to the device



*The Document Object Model (DOM) is a programming interface for HTML and XML documents. It represents the page so that programs can change the document structure, style, and content. The DOM represents the document as nodes and objects. That way, programming languages can connect to the page.

Installing NEDs

- System Installation
  - Extract the package to /var/opt/ncs/packages unless you used a non-default location
- Local Installation
  - Extract the package to the /packages subdirectory of the running directory
  - Alternatively, link to the lab-grade packages in the installation directory

```
cisco@nso:/var/opt/ncs/packages$ sudo tar -xzvf /tmp/ncs-4.1.1-cisco-ios-4.0.4.tar.gz cisco-ios
cisco-ios/
cisco-ios/build-meta-data.xml  cisco-
ios/CHANGES
cisco-ios/README
cisco-ios/netsim/
...
cisco@nso:/var/opt/ncs/packages$ sudo tar -xzvf /tmp/ncs-4.1.1-cisco-iosxr-4.1.tar.gz cisco-iosxr
cisco-iosxr/
cisco-iosxr/build-meta-data.xml  cisco-
iosxr/CHANGES
cisco-iosxr/netsim/
...
```

Using Existing Packages

- Existing packages:
  - NEDS ($NCS_DIR/packages/neds)
  - Services ($NCS_DIR/packages/services)
  - Tools ($NCS_DIR/packages/tools)
- Copy or softlink packages from the NSO installation or another location where the packages are stored:

```
cisco@nso:/var/opt/ncs/packages$ sudo cp -r $NCS_DIR/packages/neds/cisco-ios packages/
cisco@nso:/var/opt/ncs/packages$ sudo cp -r $NCS_DIR/packages/neds/cisco-iosxr packages/
```

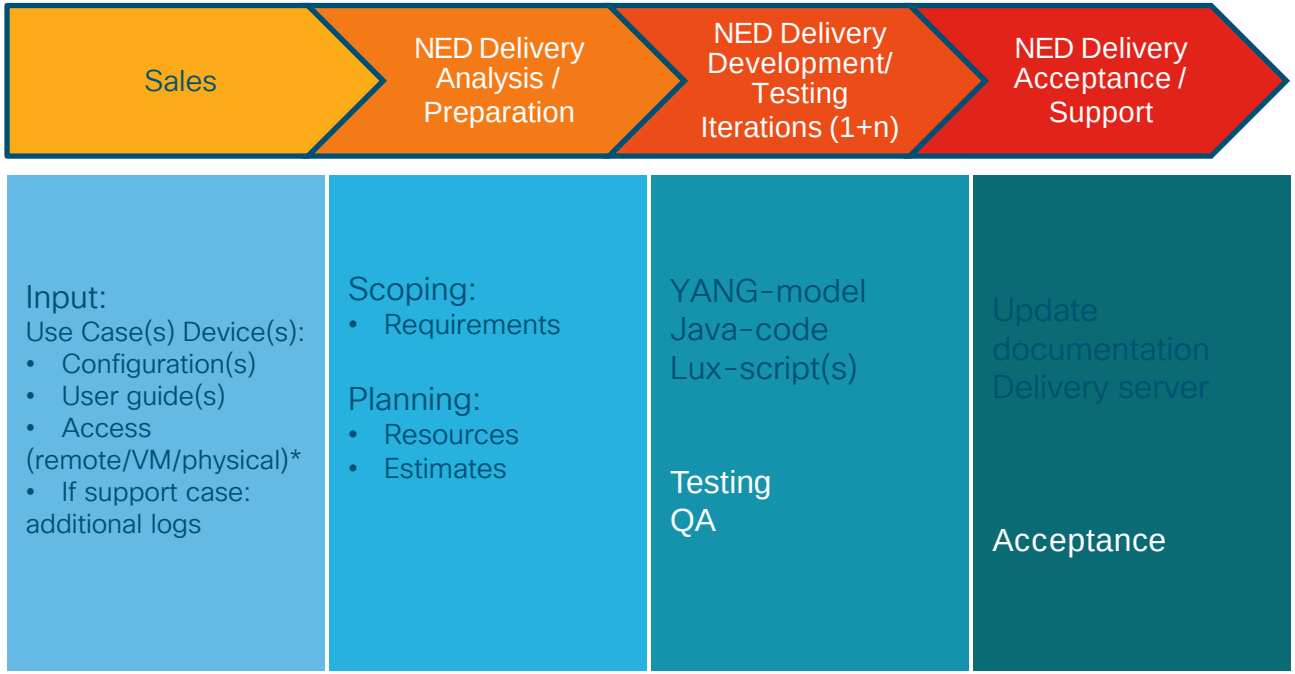
or

```
cisco@nso:/var/opt/ncs/packages$ sudo ln -s $NCS_DIR/packages/neds/cisco-iosxr packages/cisco-ios
cisco@nso:/var/opt/ncs/packages$ sudo ln -s $NCS_DIR/packages/neds/cisco-iosxr packages/cisco-iosxr
```

Activate NED Packages

- Any change in the package will not be automatically activate:
  - Recompile packages (optional; used when package was compiled using a different version of NSO)
  - Reload packages to activate them
  - Verify packages

NED Development Process



Service Package

Template Options

- The template uses tags to control the processing of lists:
  - merge (default): the changes will be merged with the existing data
  - replace: configuration will be replaced by the new configuration
  - delete: delete anything from this point
  - nocreate: merge data if it already exists, otherwise do nothing

System Requirements

System Requirements: Operating System

- Supported architecture: 64-bit x86 architecture
- Two distributions available:

  OSX (e.g. nso-4.5.2.darwin.x86_64.installer.bin)

  Linux (e.g. nso-4.5.2.linux.x86_64.installer.bin)
- Java JDK version 7.0 or higher (i.e. development version 1.7 or higher)

System Configuration and Installation Type

- NSO has two installation types that influence system administration:
  - Local installation: should be used development, testing and similar environments
  - System installation: used in a final production environment
- Understand the implications of installation types when managing the system:
  - Location of files and directories
  - Availability of certain features

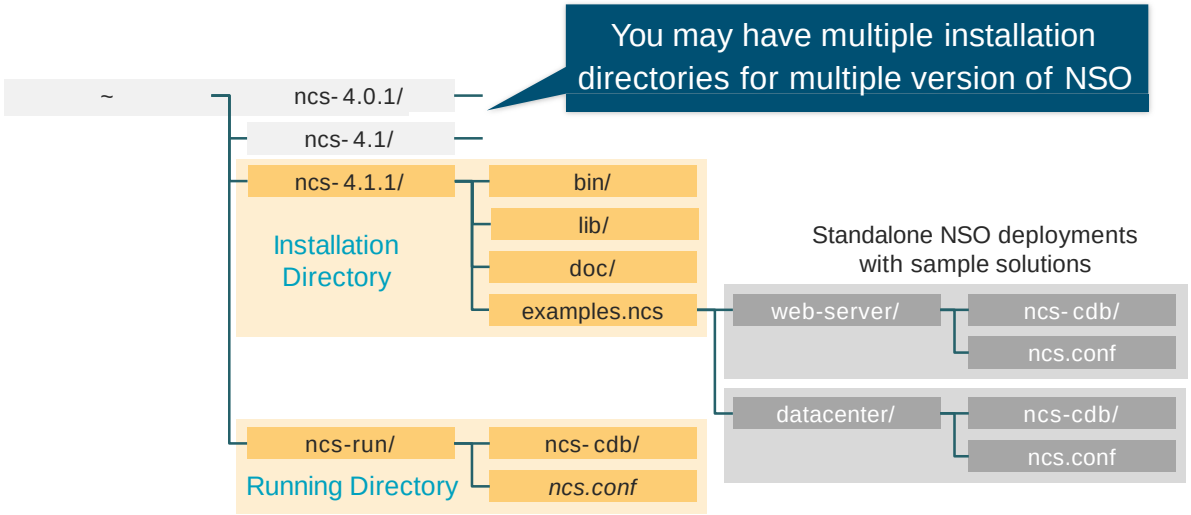
System Installation

- Intended for production environment and pre-production verification lab
- NSO can run as root or specific user (from version 4.0.1)
- System integrated with the Linux OS:
- Installation directory:/opt/ncs/ncs-4.1.1 (linked to by /opt/ncs/current)
- Running directory: /var/opt/ncs
- Log directory: /var/log/ncs
- Configuration directory: /etc/ncs

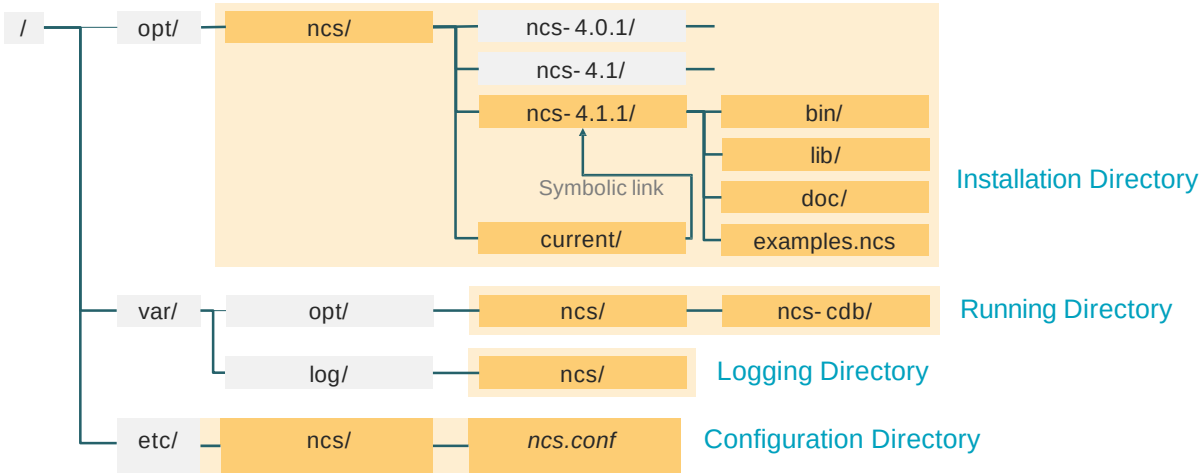
Local Installation

- Intended for in labs and on desktops
- NSO can run as user
- Simple to switch between NSO instances (versions)
- Everything installed in one or two directories
- Installation directory: all static files (e.g. binaries, libraries, scripts, tools)
- Running directory: all dynamic files (e.g. CDB, logs, configuration, packages)

Cisco NSO Local Installation Directory Structure



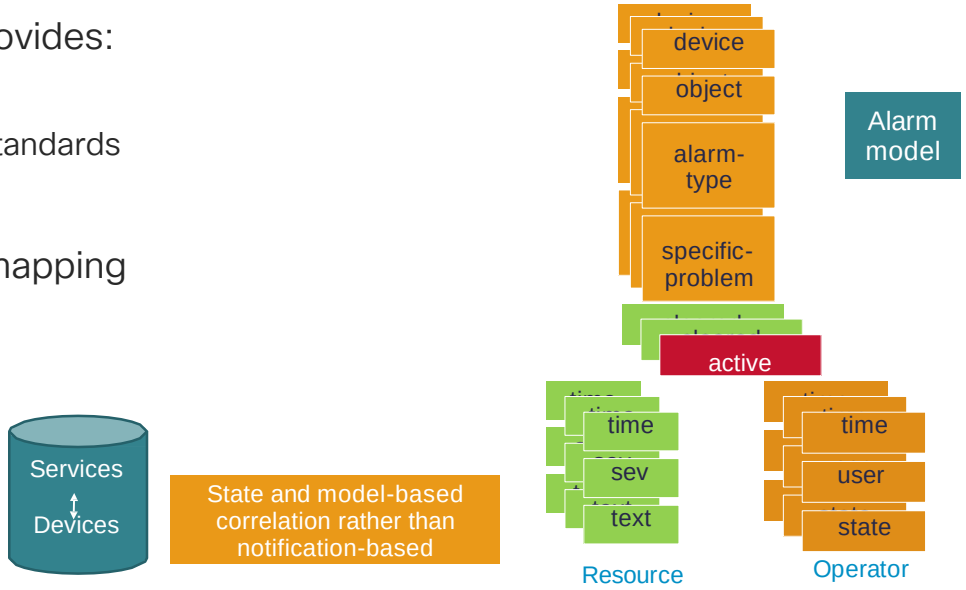
Cisco NSO System Installation Directory Structure



Other

Alarm Manager

- Manages native alarms and provides:
  - Extensible alarm-types
  - Configurable mapping to alarm standards
  - Northbound SNMP Alarm MIB
- CDB knows device – service mapping




cisco
You make possible