PROGRAMMABILITY CHALLENGES

Unstructured vs Structured Data

Wayne Rooney Forward 31 David de Gea Goalkeeper 26

Is there a hierarchy?

What are the possible fields?

Where does it begin and where does it end?

```
<Player>
  <Name>Wayne Roonery</Name>
  <Position>Forward</Position>
  <Age>31</Age>
</Player>
<Player>
  <Name>David de Gea</Name>
  <Position>Goalkeeper</Position>
  <Age>26</Age>
</Player>
```

Problem with Cisco CLI commands

SmartPro

```
Cts sxp connection peer 10.10.10.10 password default mode local listener
hold-time 3600 36000
```

- What fields correspond to what? What fields are allowed? How do I compare fields?
- When making network automation, the burden of knowing structure and syntax of the commands is placed on the developer.

Current network automation is this:

Making a machine use a human-oriented interface.

So how do can we interact with Cisco CLI in a machine oriented way?

First, we need a way to translate CLI into a structured Data Form…

INTRODUCTION TO YANG

What is Yang?

It is not the opposite of
Yin

Stands for:

Yet
Another
Next
Generation
(Data Modeling Language)

YANG is a data modeling language used to model configuration and state data manipulated by the Network Configuration Protocol (NETCONF)

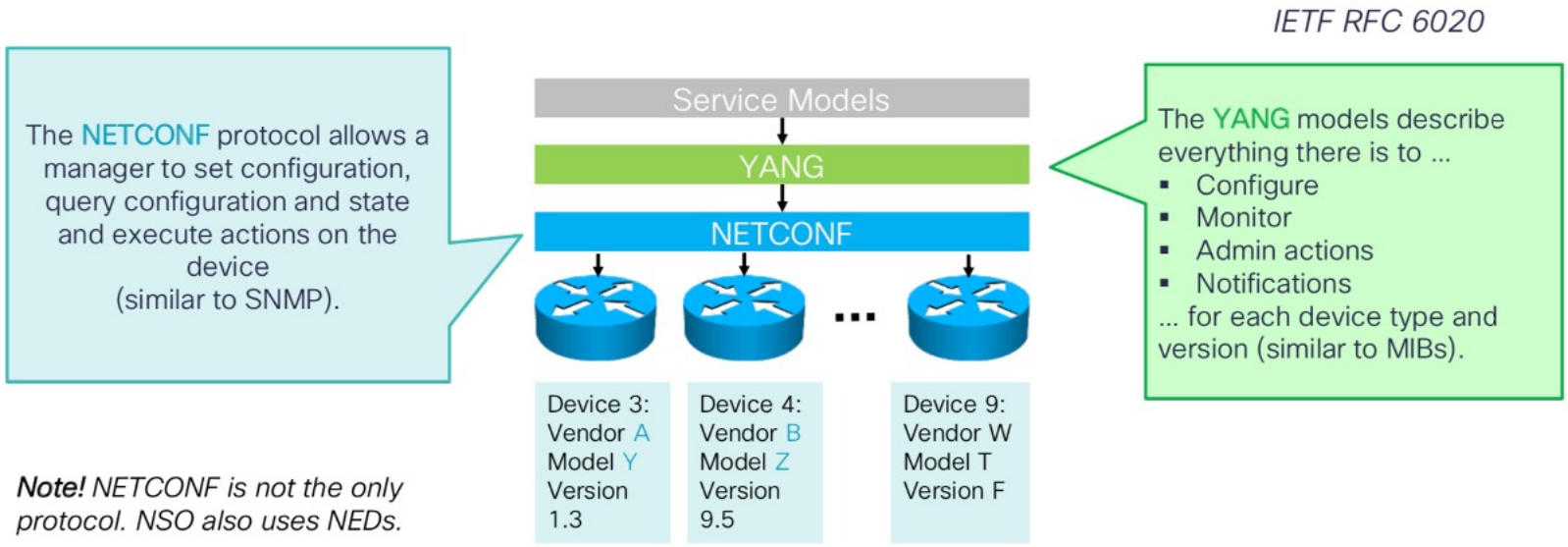
That's nice.. But what *IS* it really?

Yang allows for abstractions to be defined within the Yang language and then mapped into a markup language.

In our case, XML

What is YANG?

- YANG is a data modeling language used to model configuration, state data, and administrative actions manipulated by the NETCONF protocol. YANG was originally published as RFC 6020 in September 2010. Based on real-world user experience, the original RFC was updated to YANG 1.1 in RFC 7950 in August 2016.



Yang types and a way to think about them

- Container
  - Groups things together
- List
  - A collection of containers
- Leaf
  - A end node of data
- Leaf-List
  - A list of single items

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.

Yang

```
Container FootballTeam {
  Leaf TeamName  {type string;}
  List Player {
    Leaf PlayerName {type string;}
    Leaf Position {type string;}
    Leaf Age {type uint8;}
    }
}
```

Modeling Cisco IOS commands in Yang

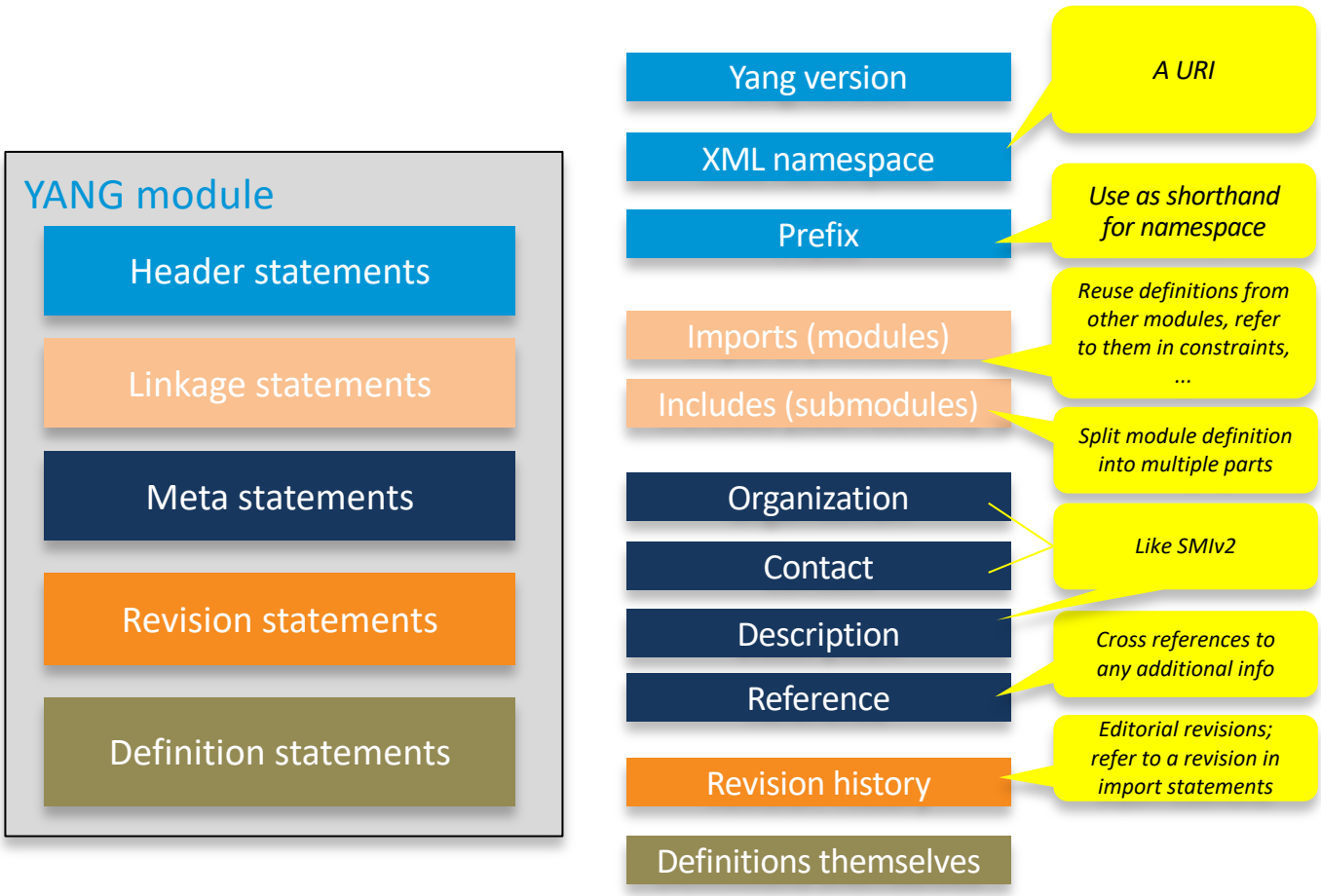
```
radius server <AAA Server>
    address ipv4 <IP Address> auth-port 1812 acct-port 1813
    key 7 <Encrypted Key>
```

```
container radius {
  list server {
    leaf id {type string;}
    container address {
      container ipv4 {
        leaf host {type string;}
        leaf auth-port {type uint16;}
        leaf acct-port {type uint16;}
        container key {
          leaf encryption {type enumeration;}
          leaf key {type string;}}}}}}
```

YANG Structure Example

```
+--rw if:interfaces
        +--rw if:interface [name]
           ...
           +--rw ipv4?
           |  +--rw enabled?             boolean
           |  +--rw ip-forwarding?       boolean
           |  +--rw address [ip]
           |  |  +--rw ip                inet:ipv4-address
           |  |  +--rw (subnet)?
           |  |     +--:(prefix-length)
           |  |     |  +--rw ip:prefix-length?   uint8
           |  |     +--:(netmask)
           |  |        +--rw ip:netmask?         inet:ipv4-address
           |  +--rw neighbor [ip]
           |     +--rw ip                inet:ipv4-address
           |     +--rw phys-address?     yang:phys-address
           +--rw ipv6?
              +--rw enabled?             boolean
              +--rw ip-forwarding?       boolean
              +--rw address [ip]
              |  +--rw ip                inet:ipv6-address
              |  +--rw prefix-length?    uint8
              +--rw neighbor [ip]
              |  +--rw ip                inet:ipv6-address
              |  +--rw phys-address?     yang:phys-address
              +--rw dup-addr-detect-transmits?   uint32
              +--rw autoconf
                 +--rw create-global-addresses?        boolean
                 +--rw create-temporary-addresses?     boolean
                 +--rw temporary-valid-lifetime?       uint32
                 +--rw temporary-preferred-lifetime?   uint32
```

YANG Module


YANG module
Header statements
Linkage statements
Meta statements
Revision statements
Definition statements
Yang version
XML namespace
Prefix
Imports (modules)
Includes (submodules)
Organization
Contact
Description
Reference
Revision history
Definitions themselves
A URI
Use as shorthand for namespace
Reuse definitions from other modules, refer to them in constraints, …
Split module definition into multiple parts
Like SMIv2
Cross references to any additional info
Editorial revisions; refer to a revision in import statements

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

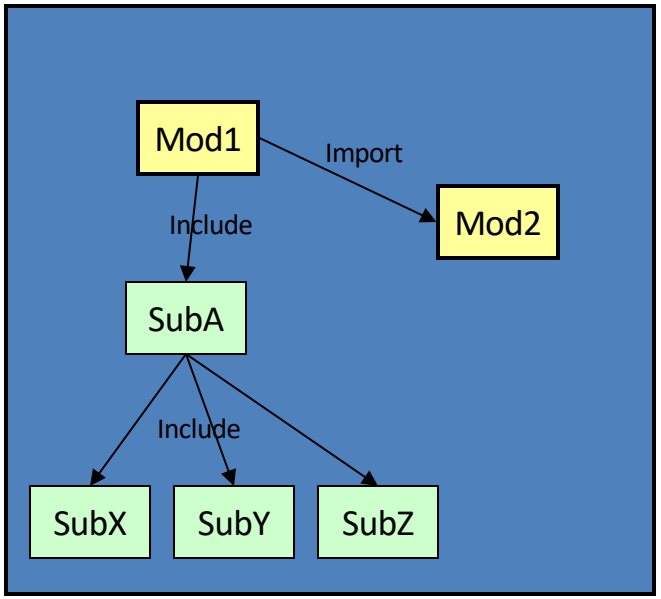
Imports & Includes

Type Definitions

Reusable Node Declarations

Modules and submodules

- Header statements
  - yang-version, namespace, prefix
- Linkage statement
  - import and include
- Meta information
  - organization, contact
- Revision history
  - revision



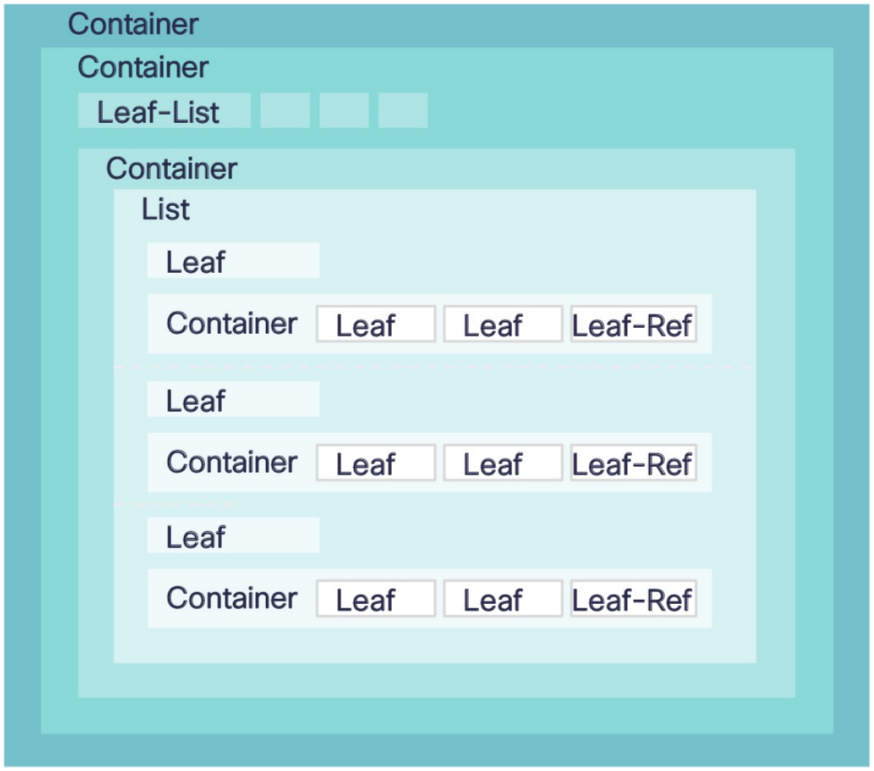
Modules and submodules

```
module acme-module {
    namespace "http://acme.example.com/module";
    prefix acme;

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

    organization "ACME Inc.";
    contact joe@acme.example.com;
    description "The module for entities
                 implementing the ACME products";

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

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)

YANG Model Statements and Hierarchy



- Leaf: single value of a defined type
- Leaf-list: multiple values of the same type
- List: multiple records containing at least one leaf (key) and an arbitrary hierarchy of other statements
- Container: groups other statements; has no value
- Leafref: is a reference to another leaf

YANG Model Statements and Hierarchy Example

```
container car {
    container v8_engine {
        leaf-list cylinder-arrangement {
          type string;
          max-elements 8;
        }
        container other-parts {
            list per-cylinder-parts {
                leaf piston-diameter {
                  type uint32;
                  range " 2000..9000" ;
                }
                container valves {
                  leaf number { ... }
                  list position { ... }
                  ...
                }
            }
        }
    }
}
```

Statement characteristics:

- Name
- Type (e.g. string, uint32)
- Constraints:
  - min-elements
  - max-elements
  - range
  - key/unique
  - leafref
  - must
  - when
- Statement content is enclosed within curly brackets
- Each sub-statement is terminated by semicolon

YANG Data Types

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

Built-in YANG Data Types

Name	Comments
binary	Any binary data
bits	A set of bits or flags
boolean	“true” or “false”
decimal64	Signed decimal number
empty	A leaf without any value
enumeration	Enumerated strings
identityref	References an abstract “identity”
Instance-identifier	Reference to another data tree node
Int8/16/32/64	Signed integer
leafref	Reference to a leaf instance
string	Human-readable string
Uint8/16/32/64	Unsigned integer
union	Choice of member types

Common YANG Data Types (RFC 6991)

IETF YANG Types

Name	Description
counter32	non-negative 32-bit integer that monotonically increases
zero-based-counter32	a counter32 that has the defined initial value zero
counter64	non-negative 64-bit integer that monotonically increases
zero-based-counter64	a counter64 that has the defined initial value zero
gauge32	non-negative integer, which may increase or decrease
gauge64	non-negative integer, which may increase or decrease
date-and-time	ISO 8601 standard for representation of dates and times
phys-address	colon-separated hexadecimal pairs (e.g. 1a:ba:da:ba:d0)
mac-address	six colon-separated hexadecimal pairs (e.g. 1a:ba:da:ba:d0:00)
xpath1.0	XPATH 1.0 expression
hex-string	colon-separated hexadecimal pairs of arbitrary length
uuid	universally unique identifier (RFC 4122)
...	

Using Types

```
import ietf-yang-types {
  prefix yang;
}
```

Common YANG Data Types (RFC 6991) (Cont.)

IETF INET Types

Name	Description
ip-version	IP protocol version: 1=IPv4, 2=IPv6, 0=unknown
dscp	Differentiated Services Code Point value: 0 to 63
ipv6-flow-label	32-bit integer in the range from 0 to 1048575
port-number	16-bit integer in the range from 0 to 65535
as-number	32-bit integer representing 2 or 4 octet BGP AS numbers
ip-address	IPv4 or IPv6 address
ipv4-address	IPv4 address (e.g. 10.1.2.3)
ipv6-address	IPv6 address (e.g. fd85:b310:6513:194b::1)
ip-prefix	IPv4 or IPv6 prefix
ipv4-prefix	IPv4 prefix (e.g. 10.1.2.0/24)
ipv6-prefix	IPv6 prefix (e.g. fd85:b310:6513:194b::/64)
domain-name	DNS domain name
host	IP address or DNS domain name
uri	uniform resource identifier
...	

Using Types

```
import ietf-inet-types {
  prefix inet;
}
```

Derived types

```
YANG Example:

typedef percent {
    type uint16 {
        range "0 .. 100";
    }
    description "Percentage";
}

leaf completed {
    type percent;
}
```

- Constraints
  - range
  - length
  - pattern
    - regex
- A modules may use types imported from other modules

```
// Weekday type
typedef weekday-type {
  type enumeration {
    enum Mon;
    enum Tue;
    enum Wed;
    enum Thu;
    enum Fri;
    enum Sat;
    enum Sun;
  }
}
```

```
NETCONF XML Encoding:

<completed>20</completed>
```

The "leaf" Statement

- A leaf has
  - one value
  - no children
  - one instance

```
YANG Example:

leaf host-name {
     type string;
     mandatory true;
     config true;
     description "Hostname for this system";
}
```

```
NETCONF XML Encoding:

<host-name>my.example.com</host-name>
```

Leaf Substatements

Substatement	Notes
Config	Config data or not (default: same as parent node, or “true” if top)
Default	Only if supplied by agent; forbidden if “Mandatory” is true
Description	Like SMIv2
If-feature	Conditional – presence of an instance depends on presence of the reference “feature” element
Mandatory	Leaf must exist in data tree (i.e., must be specified as part of config data)
Must	Any conditions on the value, must evaluate to true or data invalid, contains Xpath expression
Reference	Refers to additional info
Status	Current / deprecated / obsolete
Type	Data type of the leaf
Units	Optional, a string
When	Condition for the presence of the data item; data must be present if evaluates to true, absent otherwise; contains Xpath expression

The "leaf-list" Statement

- A leaf-list has
  - one value
  - no children
  - multiple instances

```
YANG Example:

leaf-list domain-search {
    type string;
    ordered-by user;
    description "List of domain names to search";
}
```

```
NETCONF XML Encoding:

<domain-search>high.example.com</domain-search>
<domain-search>low.example.com</domain-search>
<domain-search>everywhere.example.com</domain-search>
```

Leaf-list Substatement

Substatement	Notes
Config	Config data or not (default: same as parent node, or “true” if top)
Description	Like SMIv2
If-feature	Conditional – presence of an instance depends on presence of the reference “feature” element
Max-elements	Defaults to “unbounded”
Min-elements	Defaults to zero
Must	Any conditions on the value, must evaluate to true or data invalid, contains Xpath expression
Ordered-by	Specifies whether ordered by system (default) or by user. Important implementation implications
Reference	Refers to additional info
Status	Current / deprecated / obsolete
Type	Data type of the leaf
Units	Optional, a string
When	Data must be present if evaluates to true, absent otherwise; contains Xpath expression
	No default substatement!

The "list" Statement

```
YANG Example:

list user {
    key name;
    leaf name {
        type string;
    }
    leaf uid {
        type uint32;
    }
    leaf full-name {
        type string;
    }
    leaf class {
        type string;
        default viewer;
    }
}
```

```
NETCONF XML Enc

<user>
  <name>glocks</name>
  <full-name>Goldie</full-name>
  <class>intruder</class>
</user>
<user>
  <name>snowey</name>
  <full-name>Snow</full-name>
  <class>free-loader</class>
</user>
<user>
  <name>rzull</name>
  <full-name>Repun</full-name>
</user>
```

- A list is
  - uniquely identified by key(s)
  - holds related children
  - no value
  - multiple instances

List Substatement

Substatement	Notes
Config	Config data or not (default: same as parent node, or “true” if top)
Description	Like SMIv2
If-feature	Conditional – presence of an instance depends on presence of the reference “feature” element
key	Leaf identifiers of leaves that are part of the list and serve as keys
Max-elements	Defaults to “unbounded”
Min-elements	Defaults to zero
Must	Any conditions on the value, must evaluate to true or data invalid, contains Xpath expression
Ordered-by	Specifies whether ordered by system (default) or by user. Important implementation implications
Reference	Refers to additional info
Status	Current / deprecated / obsolete
Unique	Indicates whether combination of values (not just keys) must be unique; example: IP/Port combination
When	Data must be present if evaluates to true, absent otherwise; contains Xpath expression
	No defaults
Leaf, List, Leaf-list, Container, Choice, Grouping, Uses, anyxml	Other nodes contained underneath

The "container" Statement

```
YANG Example:

container system {
    container services {
        container ssh {
            presence "Enables SSH";
            description "SSH service specific configuration";
             // more leafs, containers and stuff here...
        }
    }
}
```

```
NETCONF XML Encoding:

<system>
  <services>
     <ssh/>
  </services>
</system>
```

- A container has
  - no value
  - holds related children
  - one instance

May have specific meaning (presence)

Or may simply contain other nodes

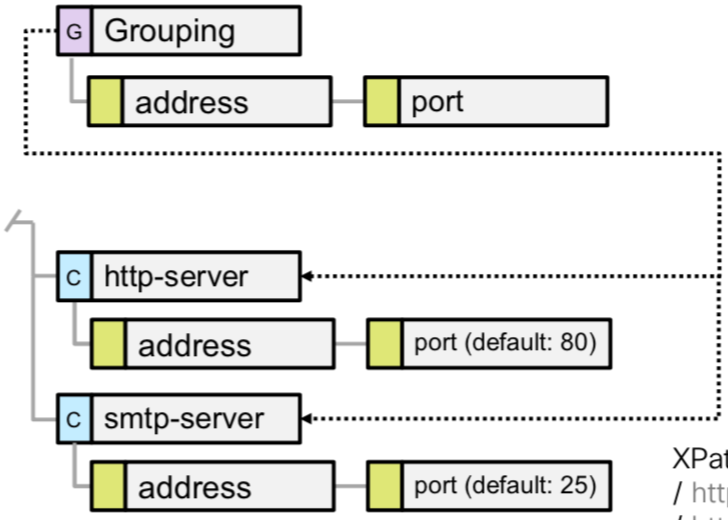
Container Substatement

Substatement	Notes
Config	Config data or not (default: same as parent node, or “true” if top)
Description	Like SMIv2
If-feature	Conditional – presence of an instance depends on presence of the reference “feature” element
Must	Any conditions on the value, must evaluate to true or data invalid, contains Xpath expression
Presence	Specifies whether presence or absence carries some meaning, i.e. whether container serves as a “configuration knob”, or if it is used only for structure
Reference	Refers to additional info
Status	Current / deprecated / obsolete
Uses	Data type of the leaf
When	Data must be present if evaluates to true, absent otherwise; contains Xpath expression
Leaf, List, Leaf-list, Container, Choice, Grouping, anyxml	Other nodes contained underneath

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;
    }
  }
}
```

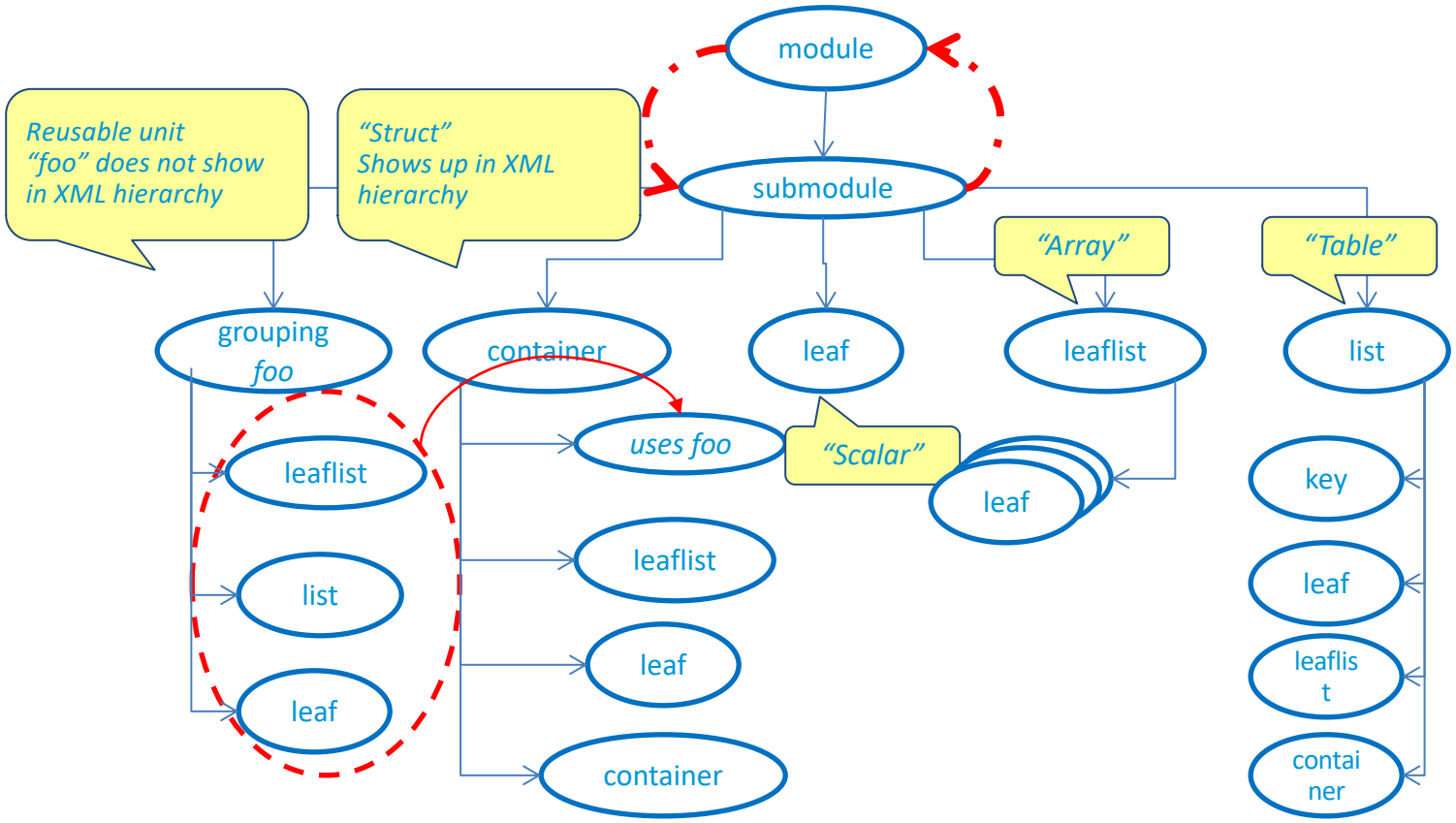


10.1.1.1	80
10.1.2.1	25

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

YANG Module Structure Summary



Tools (yang-central.org)

- pyang (python)
  - Validates YANG
  - Translates between YANG and YIN (XML)
  - Generates XSD
- yangto (binary)
  - Validates YANG
  - Generates XSD, dependencies, etc
- libsmi
  - Translates SMI/SMIv2 MIBs to YANG
- Other goodies
  - Emacs mode

YANG References

- Published as IETF RFCs

  RFC 6020: YANG – A Data Modeling Language for the Network Configuration Protocol (NETCONF)

  RFC 6021: Common YANG Data Types

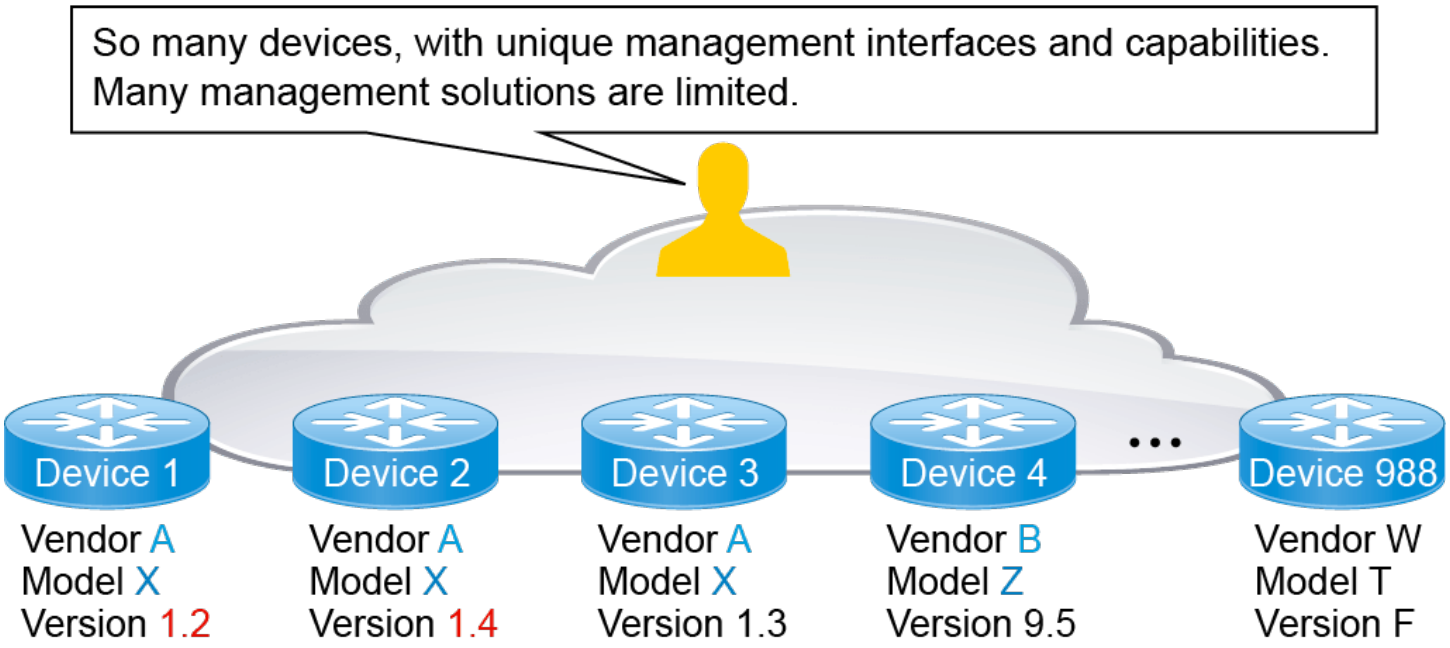
  RFC 6087: Guidelines for Authors and Reviewers of YANG Data Model Documents

  RFC 6110: Mapping YANG to Document Schema Definition Languages and Validating NETCONF Content

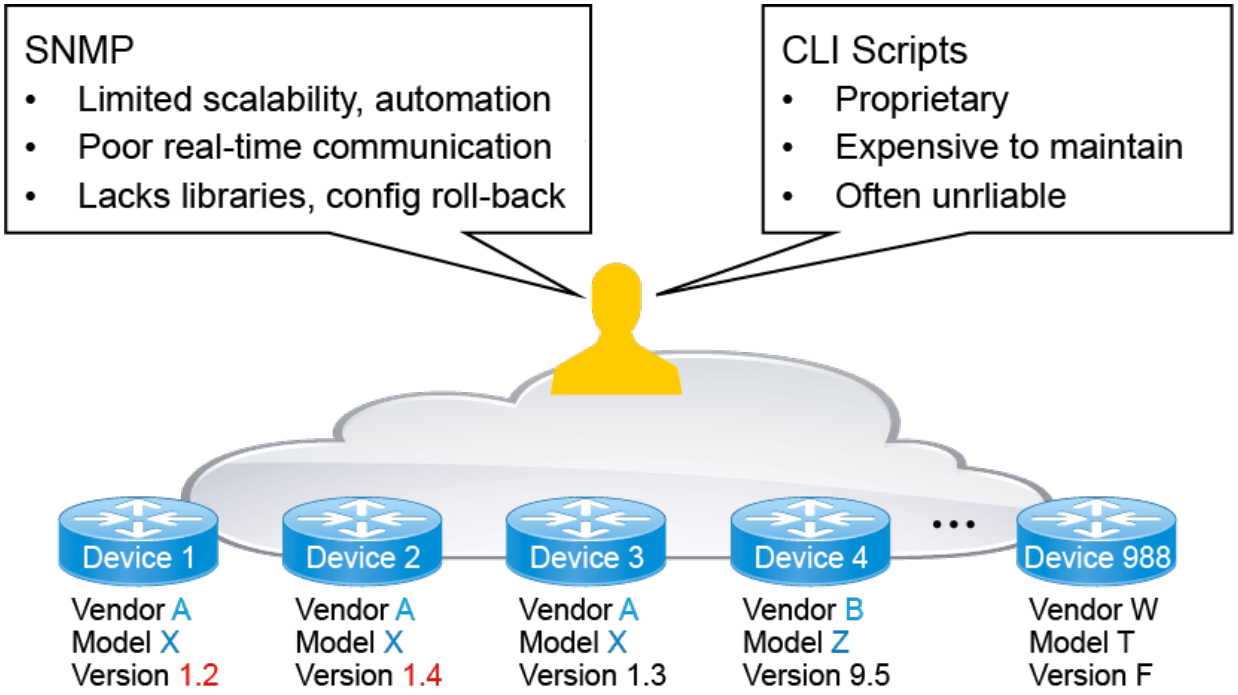
  RFC 6643:Translation of SMIv2 MIB Modules to YANG Modules

UNDERSTANDING NETCONF/RESTCONF

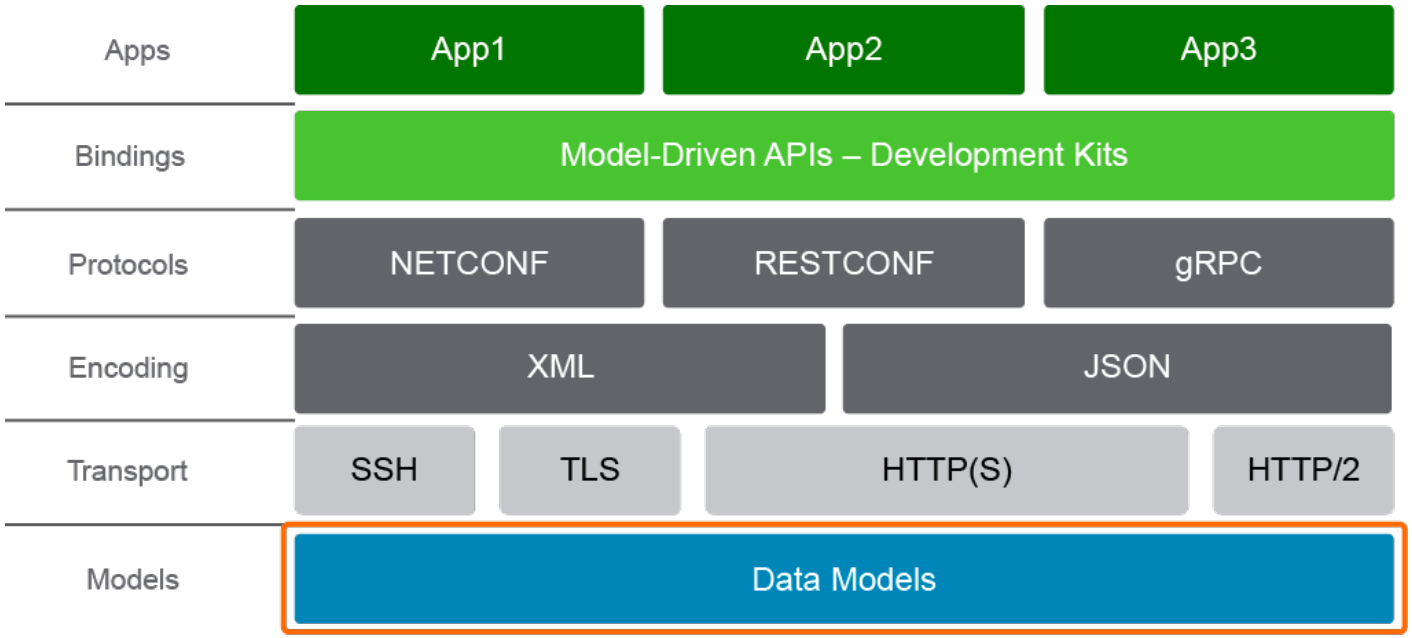
Network Management Challenges



SNMP and CLI Challenges

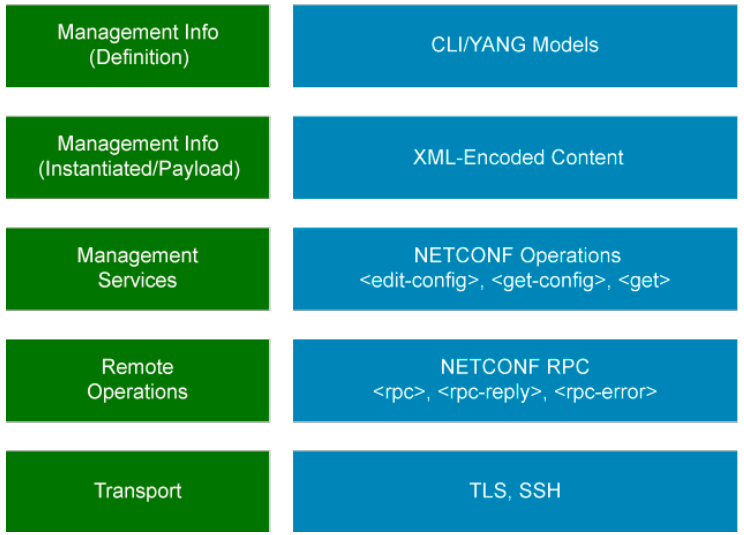


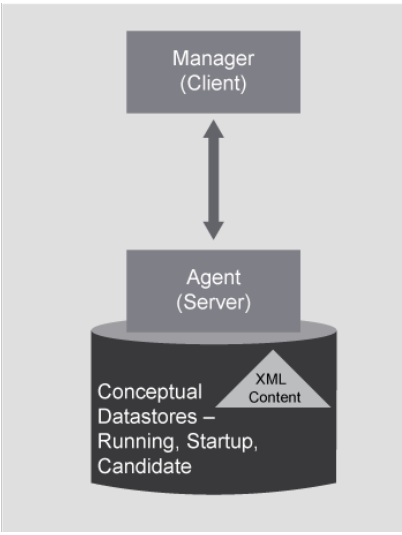
Model-Driven Programmability Stack



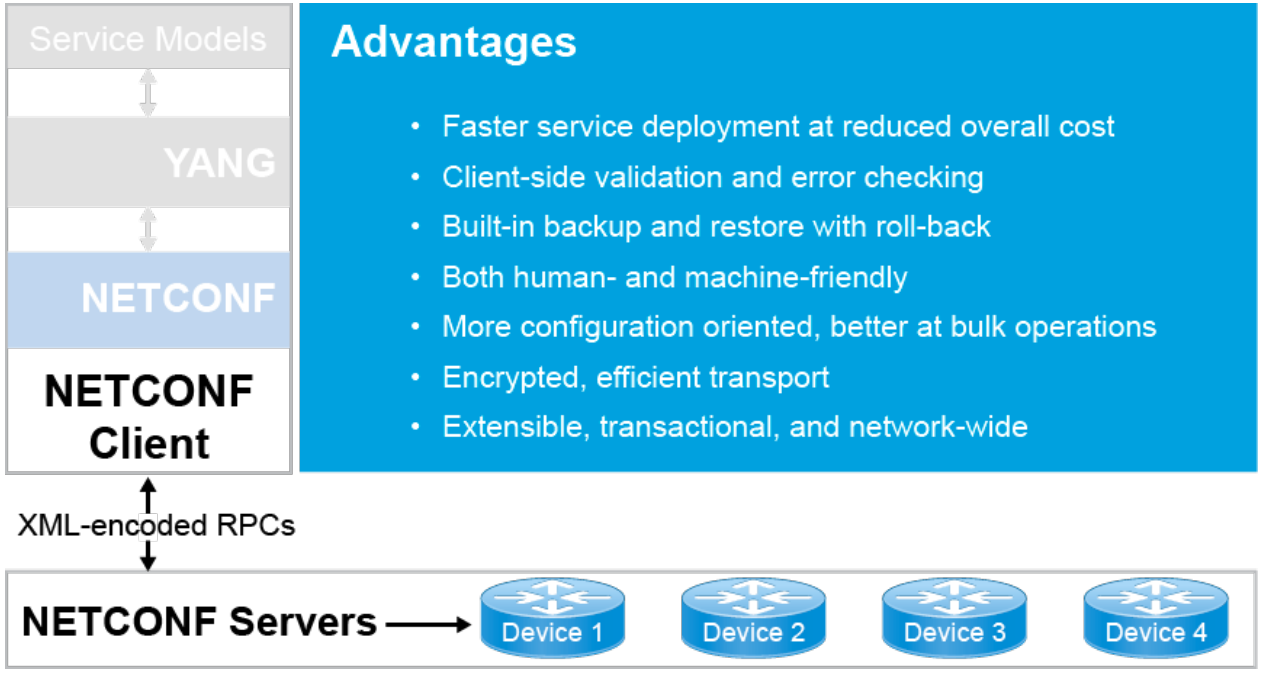
Network Automation and NETCONF

- Replaces SNMP for network management
- Manipulate configuration, retrieve operational state, and notify on events
- Initially RFC4741 updated RFC6241
- Transport over SSH(TCP830)
- Requires a NETCONF agent
- YANG was created afterwards as a common data model





NETCONF Advantages



Advantages

- Faster service deployment at reduced overall cost
- Client-side validation and error checking
- Built-in backup and restore with roll-back
- Both human- and machine-friendly
- More configuration oriented, better at bulk operations
- Encrypted, efficient transport
- Extensible, transactional, and network-wide

NETCONF Operations

Main Operations		Description
<get>	(close to 'show ?')	Retrieve running configuration and device state information
<get-config>	(close to 'show run')	Retrieve all or part of specified configuration datastore
<edit-config>	(close to 'conf t')	Loads all or part of a configuration to the specified configuration datastore

Other Operations	Description
<copy-config>	Replace an entire configuration datastore with another
<delete-config>	Delete a configuration datastore
<commit>	Copy candidate datastore to running datastore (ex: XR)
<lock> / <unlock>	Lock or unlock the entire configuration datastore system
<close-session>	Graceful termination of NETCONF session
<kill-session>	Forced termination of NETCONF session

NETCONF Datastores

Stores in NVRAM on the local device.
During initial boot, copies to running configuration.

Necessary. Controls device operation under normal circumstances
May not be directly writable.

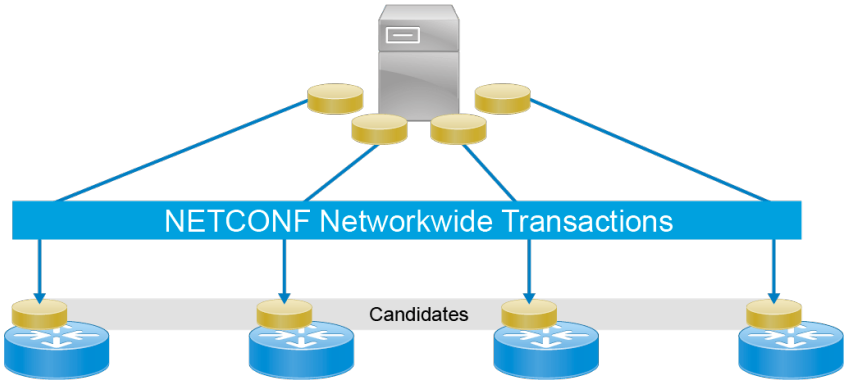
Names configuration files.
Stored locally or accessible via a URL.

Contains proposed changes in an uncommitted state.
Once committed, appears in running configuration.

NETCONF Datastores

- **running:** This data store holds the complete configuration currently active on the network device. Only one running data store can exist on a device, and it is always present. NETCONF protocol operations refer to this data store with the <running> XML element.
- **candidate:** This data store acts as a workplace for creating and manipulating configuration data. A <commit> operation causes the configuration data contained in it to be applied to the running data store.
- **startup:** This data store contains the configuration data that is loaded when the device boots up and comes online. An explicit <copy-config> operation from the <running> data store into the <startup> data store is needed to update the startup configuration with the contents of the running configuration.

Transactions and Candidate Configurations



- Once all devices report the candidate configuration as valid, the changes commit to the running configuration of all target devices. If one device reports a failure, the configuration will not commit on any device. This “all or nothing” aspect of NETCONF may seem odd at first glance. In practice, however, this feature simplifies network management processes. It also simplifies and improves the operation of NETCONF, which is based on a set of core operations and capabilities.

NETCONF Transaction and Error management

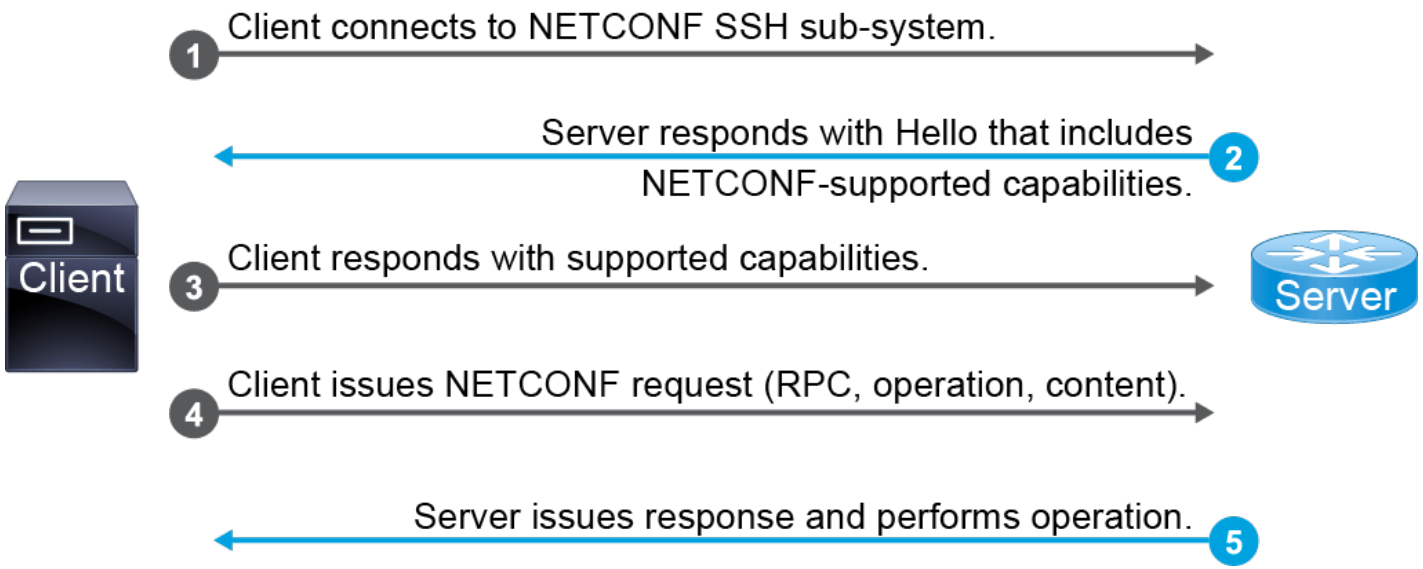
Transaction Management

- Either **all** configuration is applied **or nothing**
- Avoids **inconsistent state**
- Both at **Single Device** and **Network-wide** level

Error Management

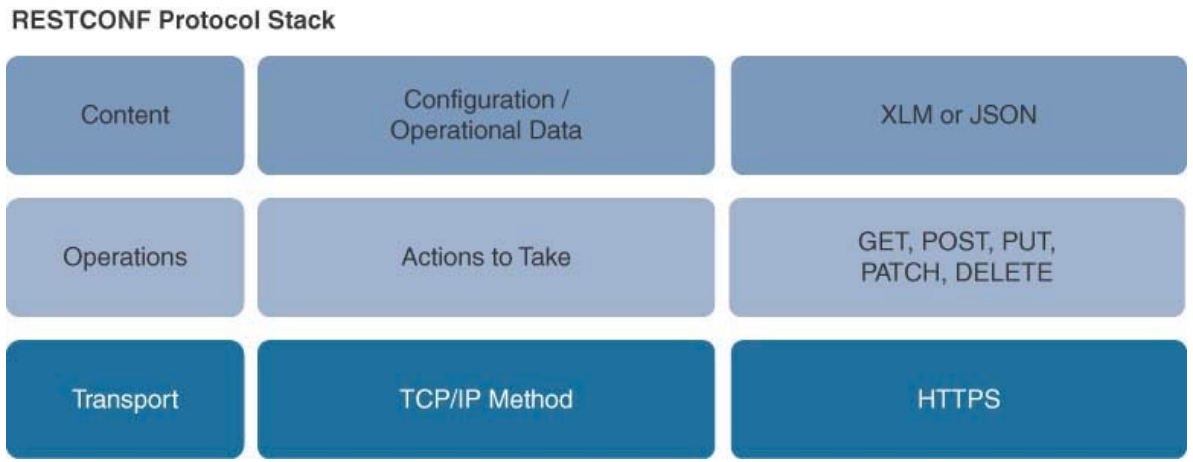
- All operations return **OK** or **error code**

NETCONF Protocol Flow

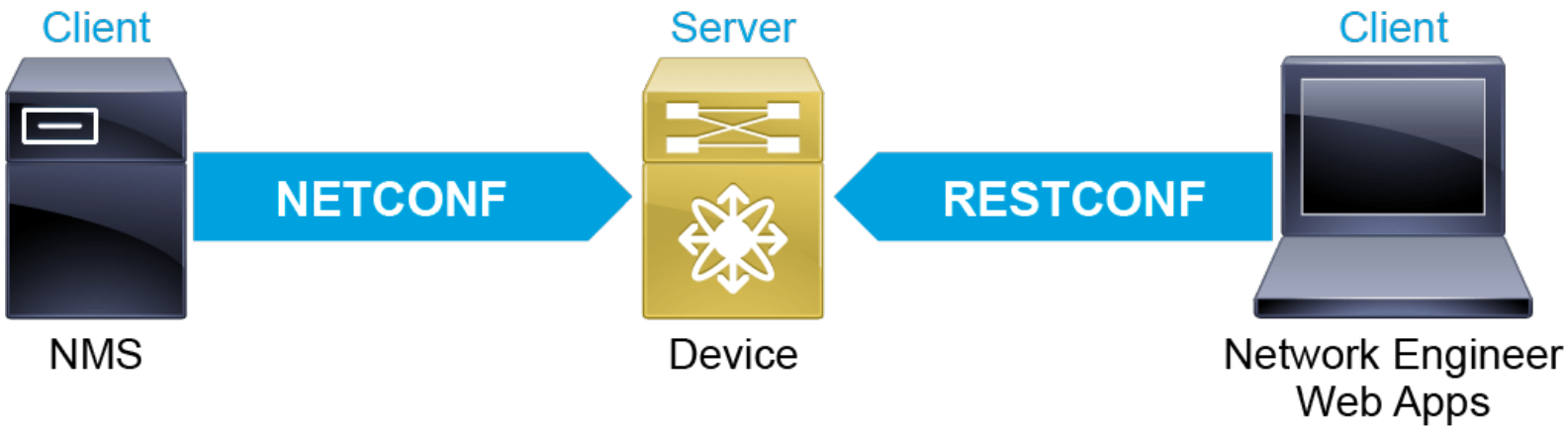


RESTCONF

- It enables a YANG model to be mapped to a RESTful interface
- Doesn't replace NETCONF
- Supports GET, POST, PUT, PATCH, and DELETE operations
- Request and respond in XML or JSON format
- HTTPS transport (TCP port 443)
- IETF Standard RFC8040



RESTCONF Versus NETCONF
Network Management



RESTCONF Versus NETCONF: Protocols