Understanding Network Programmability and Device Models

8.3.1

What is Model-Driven Programmability?

Programmability is the capability of a device to accept instructions to control or alter behavior. For network devices, this frequently means being configured and managed using software protocols. Unlike human-oriented interfaces such as CLI and GUIs, programmable interfaces are explicitly designed to be consumed by machines.

Model-driven programmability inherits the power of models, matching a device's abilities and services to standardized models. This makes it easier to configure network devices and overcome drawbacks posed by traditional network device management techniques.

What is a data model?

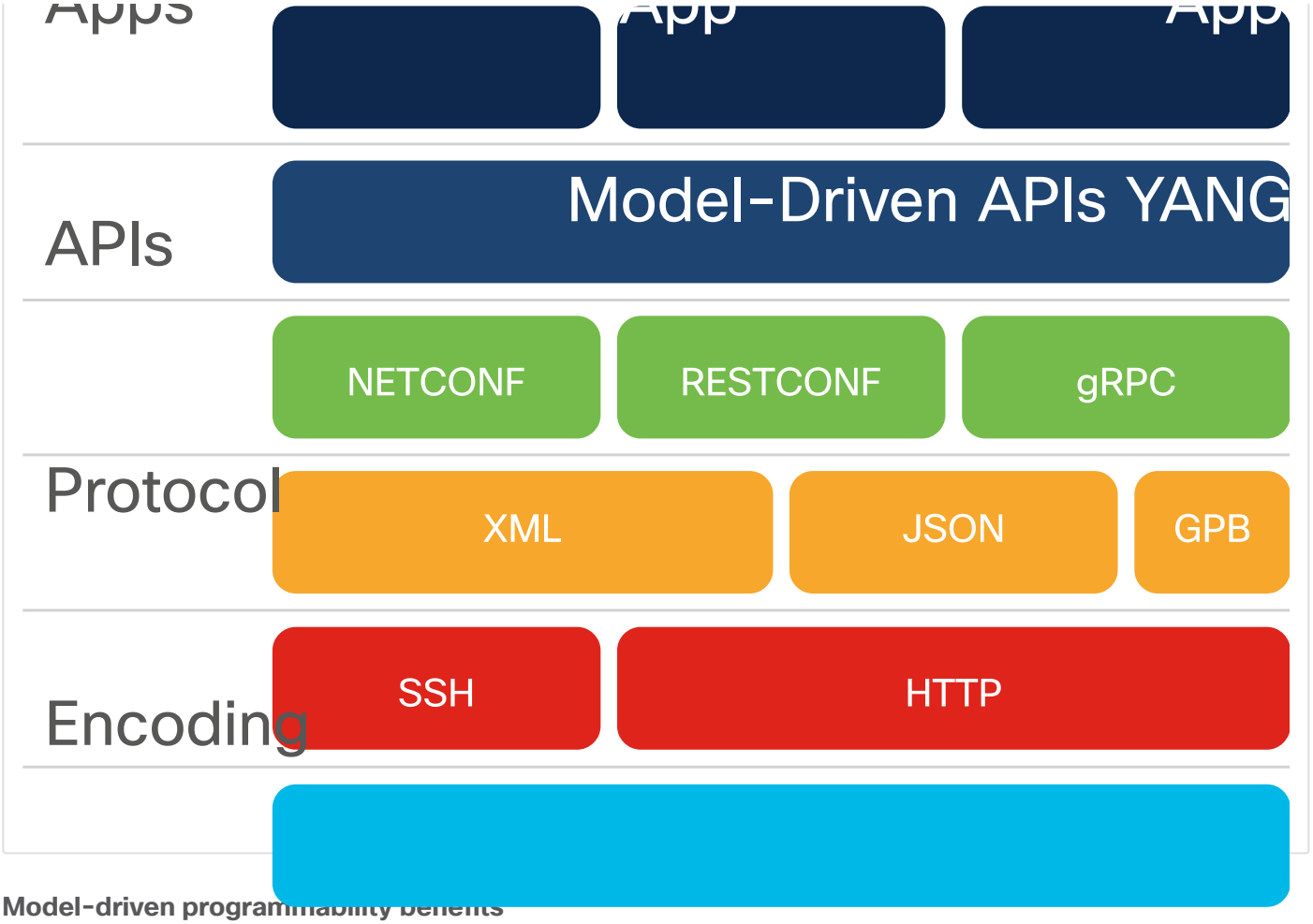
In general, a data model is a structured method to describe any object, whether it's a person, a car, a building, or some other object. For example, the personal data on a passport or driver's license can describe a person in an individual way, so those are both "data models".

In networking, "data models" are a programmatic and standards-based way of writing configurations for any network device. It can replace manual configuration, implementing YANG as the de-facto data modeling language.

In the current networking world, one set of data models is based on YANG and uses the YANG modeling language. YANG version 1 [RFC6020] or YANG version 1.1 [RFC7950] are used to specify the configuration and operational state that a device supports.

The YANG based Model-Driven Programmability Stack looks like this:

Model-Driven Programmability Stack



Model-driven programmability benefits

Traditionally, CLIs or SDK library functions are used to configure network devices. CLIs and scripts are almost always device-specific; they can only be used in the same type of devices that use the same CLIs from the same vendor. Moreover, those functions may not exist for all configuration commands, and where they do exist, it may be difficult to program a complex configuration.

However, model-driven programming provides a standard way to describe the desired configuration of the network devices.

Models

Of course, the target device itself has to support the model-driven configuration. For YANG based models, it needs to support YANG and understand higher-level protocols based on YANG, such as NETCONF and RESTCONF.

Model-based configurations still must be delivered to the network device. Model delivery might be encapsulated in an API, or handled in an SDK wrapper.

In summary, model-driven programmability:

- Provides configuration language that is human-readable
- Is Model-based, structured, and computer-friendly
- Includes support for multiple model types, including native, OpenConfig, and IETF
- Uses specification that is decoupled from transport, protocol end encoding
- Uses model-driven APIs for abstraction and simplification
- Leverages open-source and enjoys wide support

In order to understand model-driven programmability, we need to understand its key components:

- YANG
- NETCONF
- RESTCONF

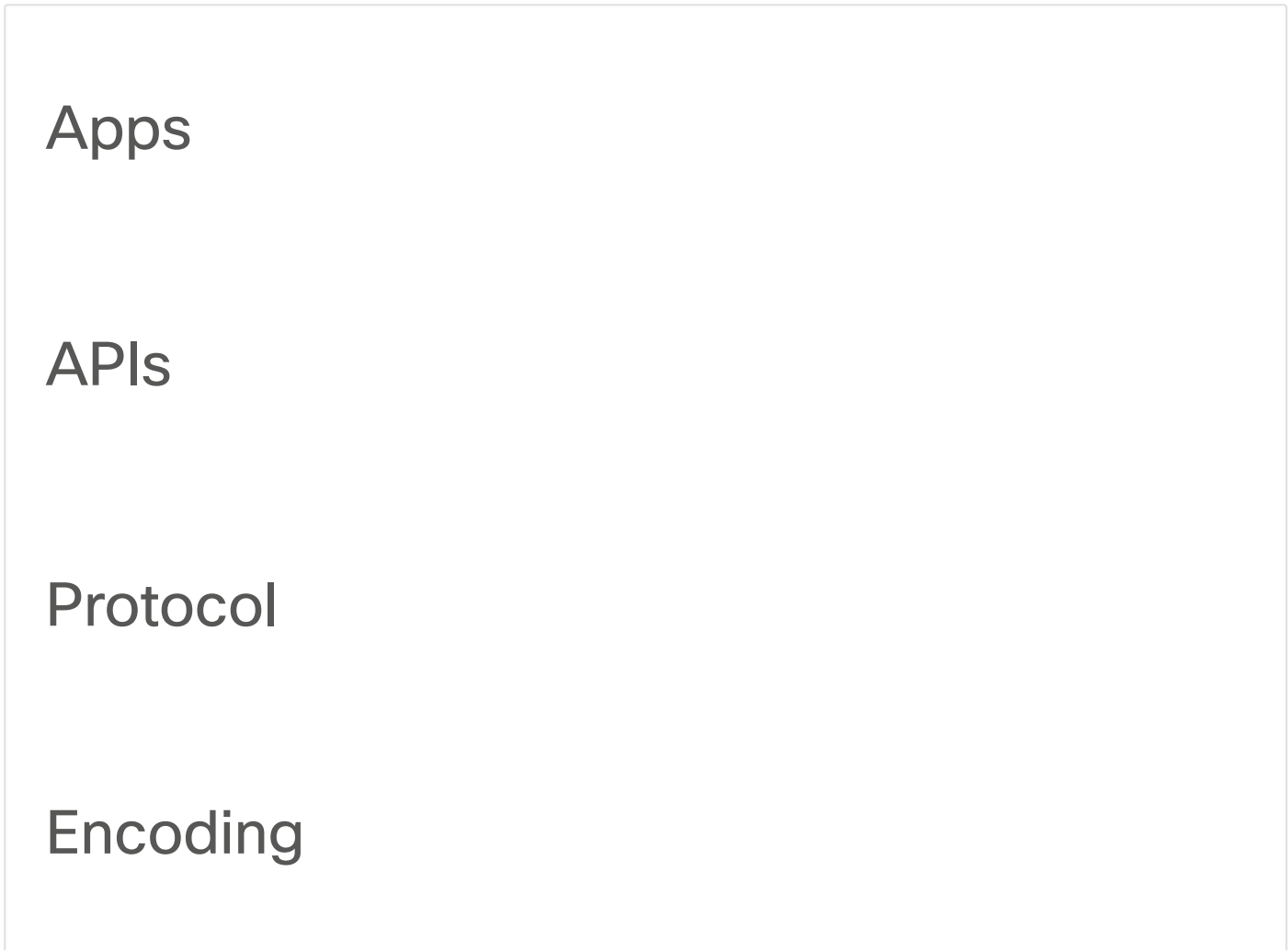
YANG is a modeling language. NETCONF and RESTCONF are protocols used for data model programmable interfaces.

8.3.2

What is YANG?

YANG, an acronym for Yet Another Next Generation, as defined in RFC7519, 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 in the Model-Driven Programmability Stack



A YANG module defines hierarchies of data that can be used for NETCONF-based operations, including configuration, state data, RPCs, and notifications. This allows a complete description of all data sent between a NETCONF client and server. YANG can also be used with protocols other than NETCONF.

Although YANG can describe any data model, it was originally designed for networking data models.

In the real world there are two types of YANG models, open and native.

- Open YANG Models: Developed by vendors and standards bodies, such as IETF, ITU, OpenConfig, etc.. They are designed to be independent of the underlying platform and normalize the per-vendor configuration of network devices.
- Native Models: Developed by vendors, such as Cisco. They relate and are designed to integrate features or configuration only relevant to that platform.

Why we need YANG for device modeling?

YANG provides a standard while still allowing for further description. We need a standard way to model network device configuration. YANG allows different network device vendors to describe their device type, configuration, and state to map to the device operation in programmatic way.

The terms used in YANG that you need to understand before you

- **anyxml**: A data node that can contain an unknown chunk of XML data.
- **augment**: Adds new schema nodes to a previously defined schema node.
- **container**: An interior data node that exists in, at most, one instance in the data tree. A container has no value, but rather a set of child nodes.
- **data model**: A data model describes how data is represented and accessed.
- **data node**: A node in the schema tree that can be instantiated in a data tree. One of container, leaf, leaf-list, list, and anyxml.
- **data tree**: The instantiated tree of configuration and state data on a device.
- **derived type**: A type that is derived from a built-in type (such as uint32), or another derived type.
- **grouping**: A reusable set of schema nodes. Grouping may be used locally in the module, in modules that include it, and by other modules that import from it. The grouping statement is not a data definition statement and, as such, does not define any nodes in the schema tree.
- **identifier**: Used to identify different kinds of YANG items by name.
- **leaf**: A data node that exists in at most one instance in the data tree. A leaf has a value but no child nodes.
- **leaf-list**: Like the leaf node but defines a set of uniquely identifiable nodes rather than a single node. Each node has a value but no child nodes.
- **list**: An interior data node that may exist in multiple instances in the data tree. A list has no value, but rather a set of child nodes.
- **module**: A YANG module defines a hierarchy of nodes that can be used for NETCONF-based operations. With its definitions and the definitions it imports or includes from elsewhere, a module is self-contained and “compilable”.
- **RPC**: A Remote Procedure Call, as used within the NETCONF protocol.

- **state data**: The additional data on a system that is not configuration data, such as read-only status information and collected statistics [RFC4741].

YANG defines four types of nodes for data modeling. The detail is described in RFC6020 section 4.2.2 or RFC 7950 section 4.2.2.

- Leaf Nodes
- Leaf-List Nodes
- Container Nodes
- List Nodes

A YANG module contains a sequence of statements. Each statement starts with a keyword, followed by zero or one argument, followed either by a semicolon (";") or a block of sub-statements enclosed within braces `("{ }")`.

```
statement = keyword [argument] (";" / "{" *statement "}")
```

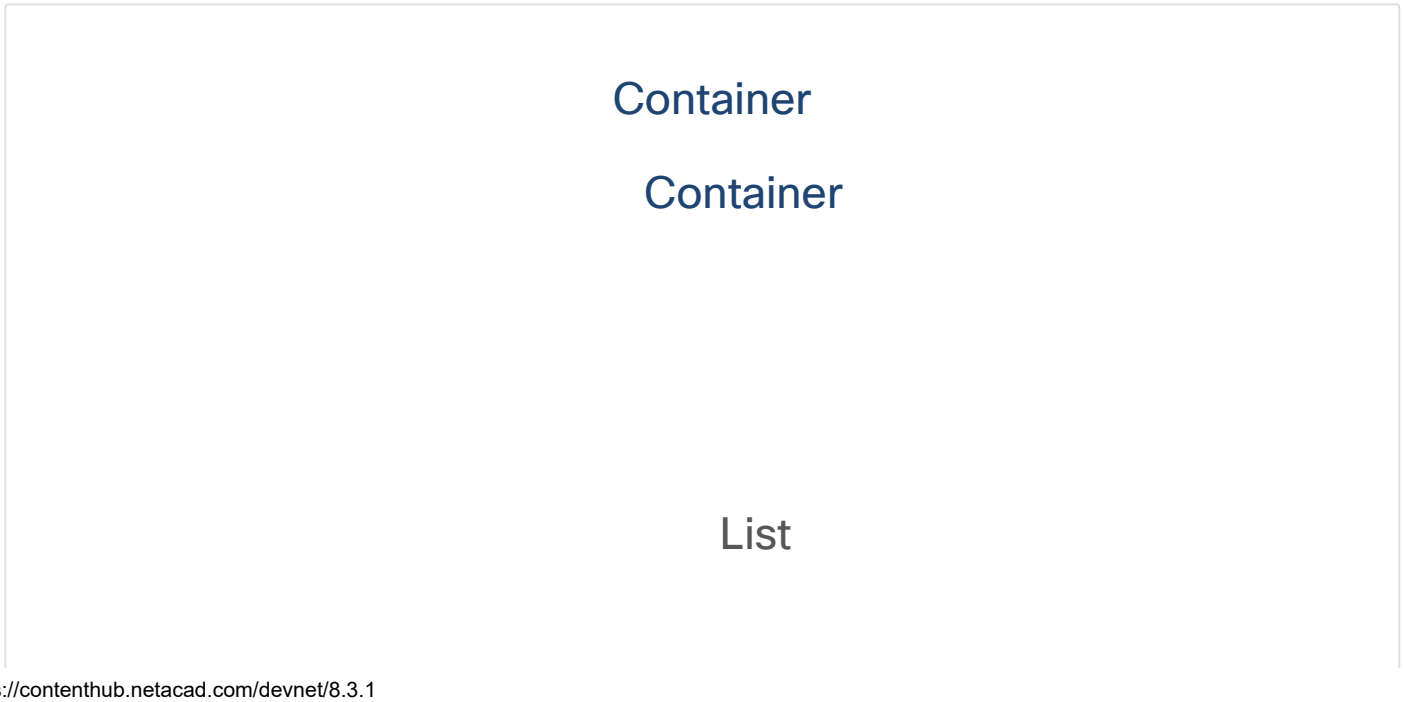
There are four major statements in a YANG module:

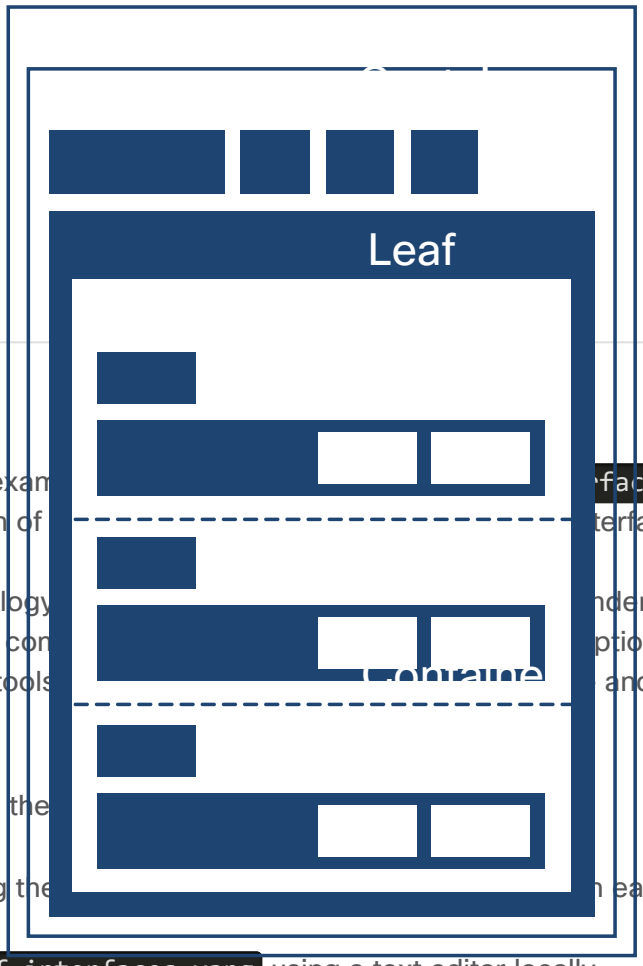
- **Container**: A grouping of other statements (has no "Value").
- **List**: Multiple records containing at least one Leaf "Key" and an arbitrary hierarchy of other statements
- **Leaf**: Single key/value pair.
- **Leaf-list**: Multiple key/values pair of the same type.

YANG Module Constructs

Modules usually root from one or more containers, and the schema tree extends to another level of data structures. A branch ends with leaves and/or leaf-lists.

YANG Module Structure





YANG in action

Now let's take a look at an exam `faces.yang`. This YANG open module contains a collection of terfaces.

When you know the terminology nderstand the content of a YANG file, which has very detailed con ptions also make the file very long. Fortunately, there are tools and concise way, and the `pyang` tool is one of them.

You can install `pyang` using the

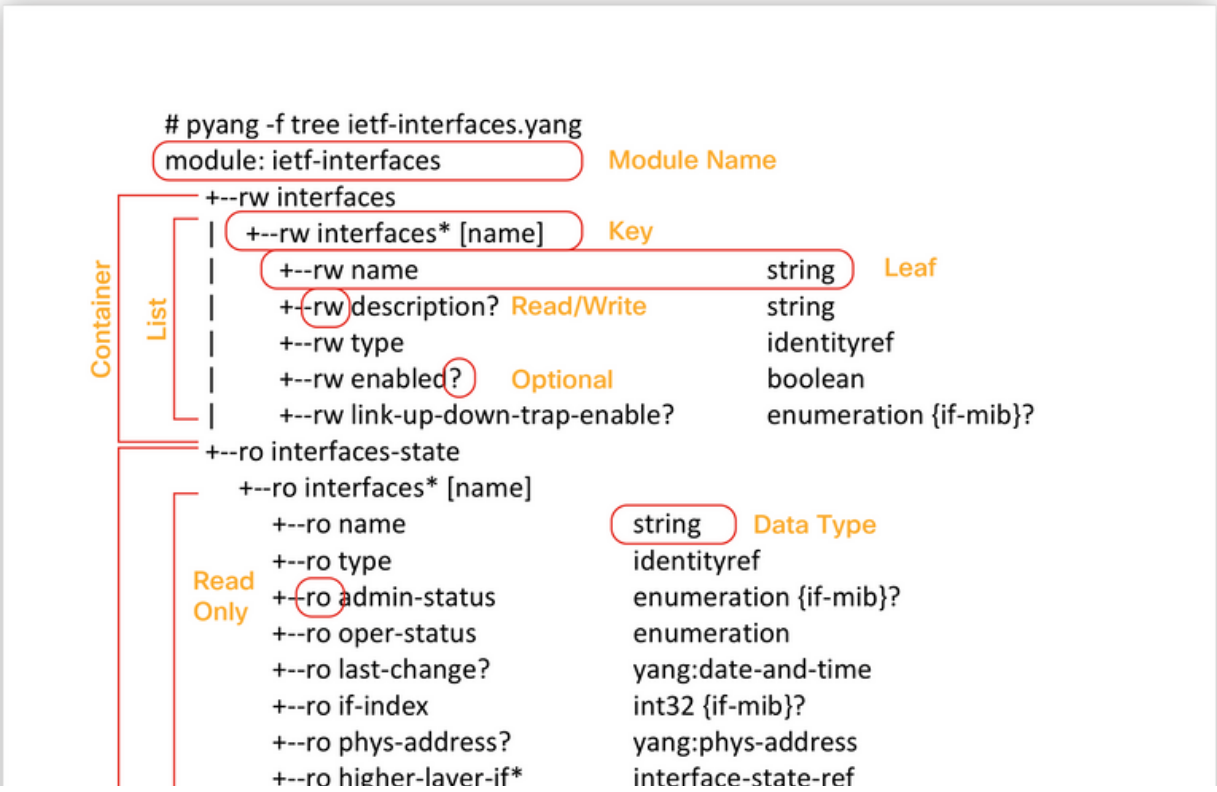
As you can see below, using the easy-to-follow tree structure.

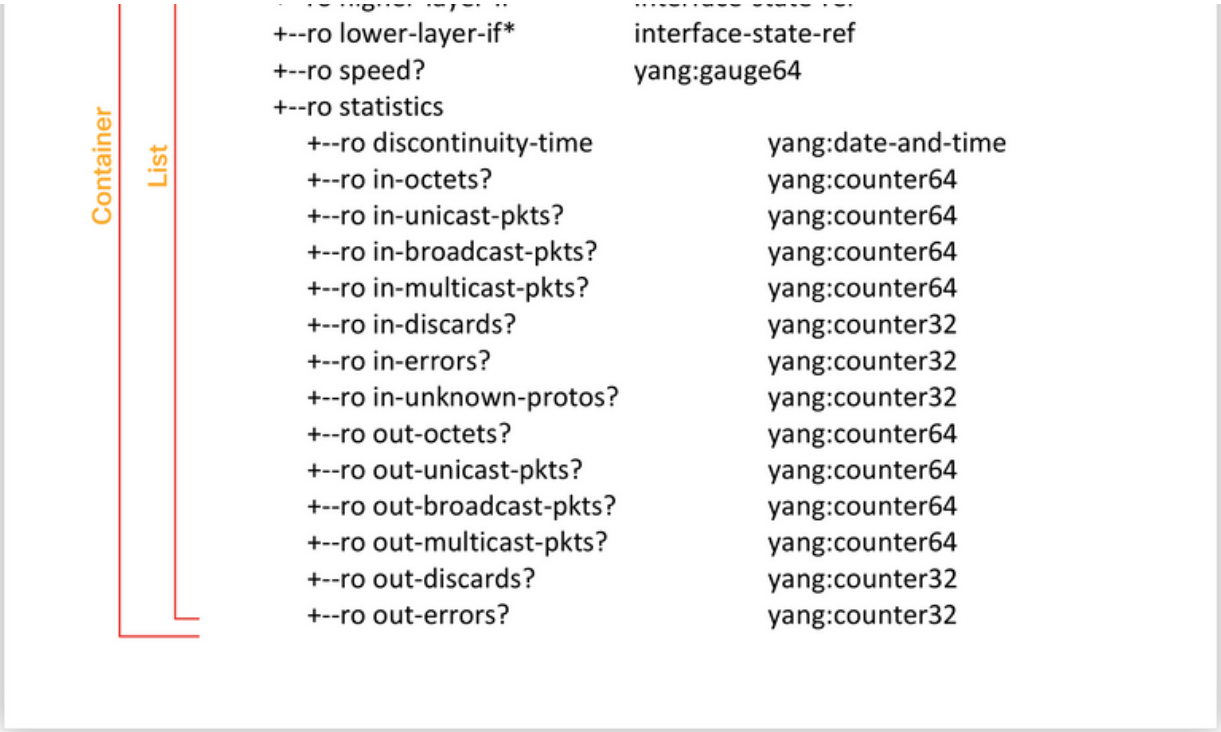
1. Create a file named `ietf-interfaces.yang` using a text editor locally.
2. Copy an example YANG file from GitHub. For example, you can go to https://github.com/YangModels/yang/blob/master/standard/ietf/RFC/ietf-interfaces%402018-02-20.yang
   for an open model, or
   https://github.com/YangModels/yang/blob/master/vendor/cisco/xr/602/ietf-interfaces.yang for a vendor-specific native model.
3. Click the **Raw** button and copy the text from the page into the local `ietf-interfaces.yang` file in a text editor.
4. Save the `ietf-interfaces.yang` file locally where you have installed `pyang`.
5. In the directory where you have stored the YANG file, run the `pyang -f tree ietf-interfaces.yang` command to generate a *tree-formatted* list of ietf-interface resources.

Pyang output example

```
module: ietf-interfaces
   +--rw interfaces
   |  +--rw interface* [name]
   |     +--rw name                        string
   |     +--rw description?                string
   |     +--rw type                        identityref
   |     +--rw enabled?                    boolean
   |     +--rw link-up-down-trap-enable?   enumeration {if-mib}?
   +--ro interfaces-state
```

```
      +--ro interface* [name]
         +--ro name                string
         +--ro type                identityref
         +--ro admin-status        enumeration {if-mib}?
         +--ro oper-status         enumeration
         +--ro last-change?        yang:date-and-time
         +--ro if-index            int32 {if-mib}?
         +--ro phys-address?       yang:phys-address
         +--ro higher-layer-if*    interface-state-ref
         +--ro lower-layer-if*     interface-state-ref
         +--ro speed?              yang:gauge64
         +--ro statistics
            +--ro discontinuity-time    yang:date-and-time
            +--ro in-octets?            yang:counter64
            +--ro in-unicast-pkts?      yang:counter64
            +--ro in-broadcast-pkts?    yang:counter64
            +--ro in-multicast-pkts?    yang:counter64
            +--ro in-discards?          yang:counter32
            +--ro in-errors?            yang:counter32
            +--ro in-unknown-protos?    yang:counter32
            +--ro out-octets?           yang:counter64
            +--ro out-unicast-pkts?     yang:counter64
            +--ro out-broadcast-pkts?   yang:counter64
            +--ro out-multicast-pkts?   yang:counter64
            +--ro out-discards?         yang:counter32
            +--ro out-errors?           yang:counter32
```





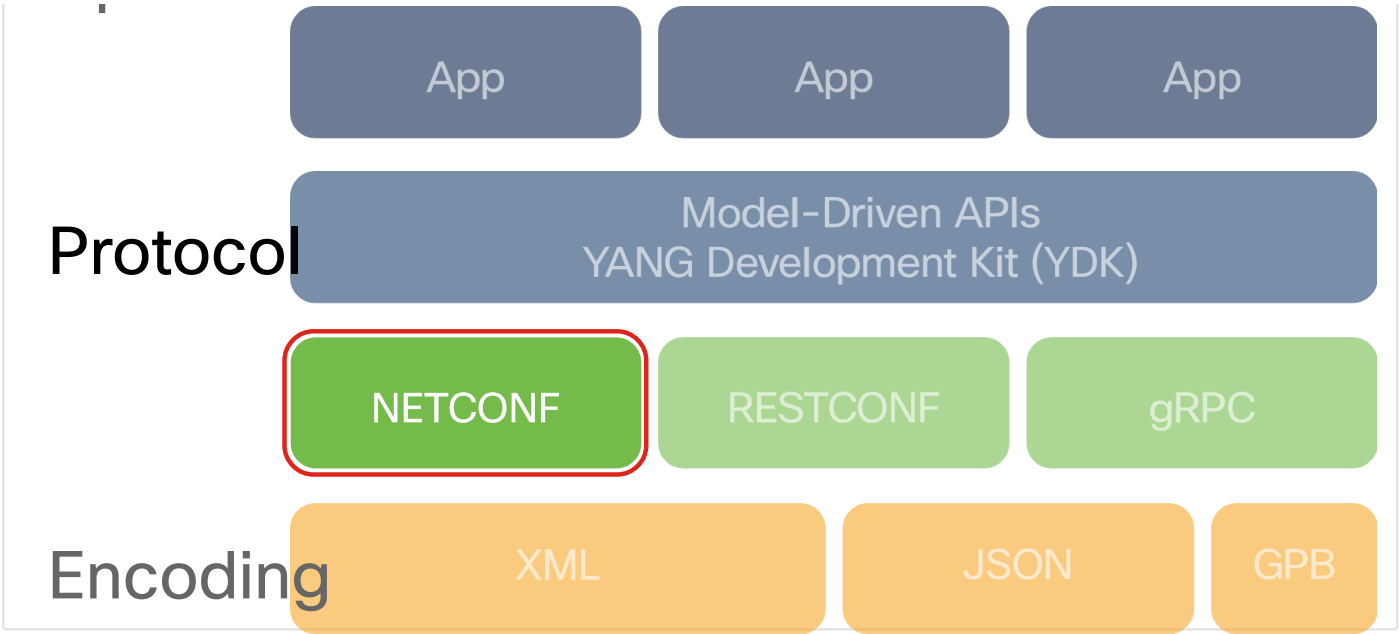
Since YANG was first used with NETCONF, take a closer look at NETCONF and see how the two relate.

8.3.3

What is NETCONF?

Network Configuration (NETCONF) is a protocol defined by the IETF RFC7519. It is designed to install, manipulate, and delete the configuration of network devices. It is the primary protocol used with YANG data models today. Its operations are realized on top of Remote Procedure Call (RPC) exchanged via a secure transport protocol such as SSH.

NETCONF in the Model-Driven Programmability Stack



The NETCONF protocol uses XML-based data encoding for both the configuration data and the protocol messages.

The NETCONF protocol provides a small set of operations to manage device configurations and retrieve device state information. The base protocol provides operations to retrieve, configure, copy, and delete configuration data stores.

NETCONF Protocol Operations

The NETCONF protocol provides a set of operations to manage device configurations and retrieve device state information. The base protocol includes the following protocol operations:

Operation	Description
get	retrieve running configuration and device state information.
get-config	retrieve all or part of a specified configuration.
edit-config	edit a device configuration.
copy-config	create or replace an entire configuration datastore with another complete configuration datastore.
delete-config	delete a configuration in a data store.
lock	lock the entire configuration datastore system of a device.
unlock	release a configuration lock, previously obtained with the **lock** operation.
close-session	request graceful termination of a NETCONF session.
kill-session	force termination of a NETCONF session.

NETCONF versus SNMP

NETCONF and SNMP protocols are both defined to remotely configure devices.

Feature comparison

SNMP:

- Uses pull model when retrieving data from device which does not scale up well for high-density platforms
- Does not have a discovery process for finding Management Information Base (MIBs) supported by a device
- Does not support the concept of transactions
- Lacks backup-and-restore of element configuration
- Limited industry support for configuration MIBs

NETCONF was designed to enable:

- Multiple configuration data stores (candidate, running, startup)
- Device-level and network-wide transactions
- Configuration testing and validation
- Distinction between configuration and operational data
- Selective data retrieval with filtering
- Streaming and playback of event notifications
- Extensible remote procedure calls
- Built-in capability exchange

The NETCONF RFC has three distinct data stores that are the target of configuration reads and writes. These are:

- running (mandatory)
- candidate (optional)
- startup (optional)

NETCONF versus SNMP capabilities

Use Case	NETCONF	SNMP
Get collection of status fields	Yes	Yes
Set collection of configuration fields	Yes	Yes
Set configuration fields in transaction	Yes	No
Transactions across multiple network elements	Yes	No
Send event notifications	Yes	Yes
Secure protocol	Yes	Yes
Test configuration before final commit	Yes	No

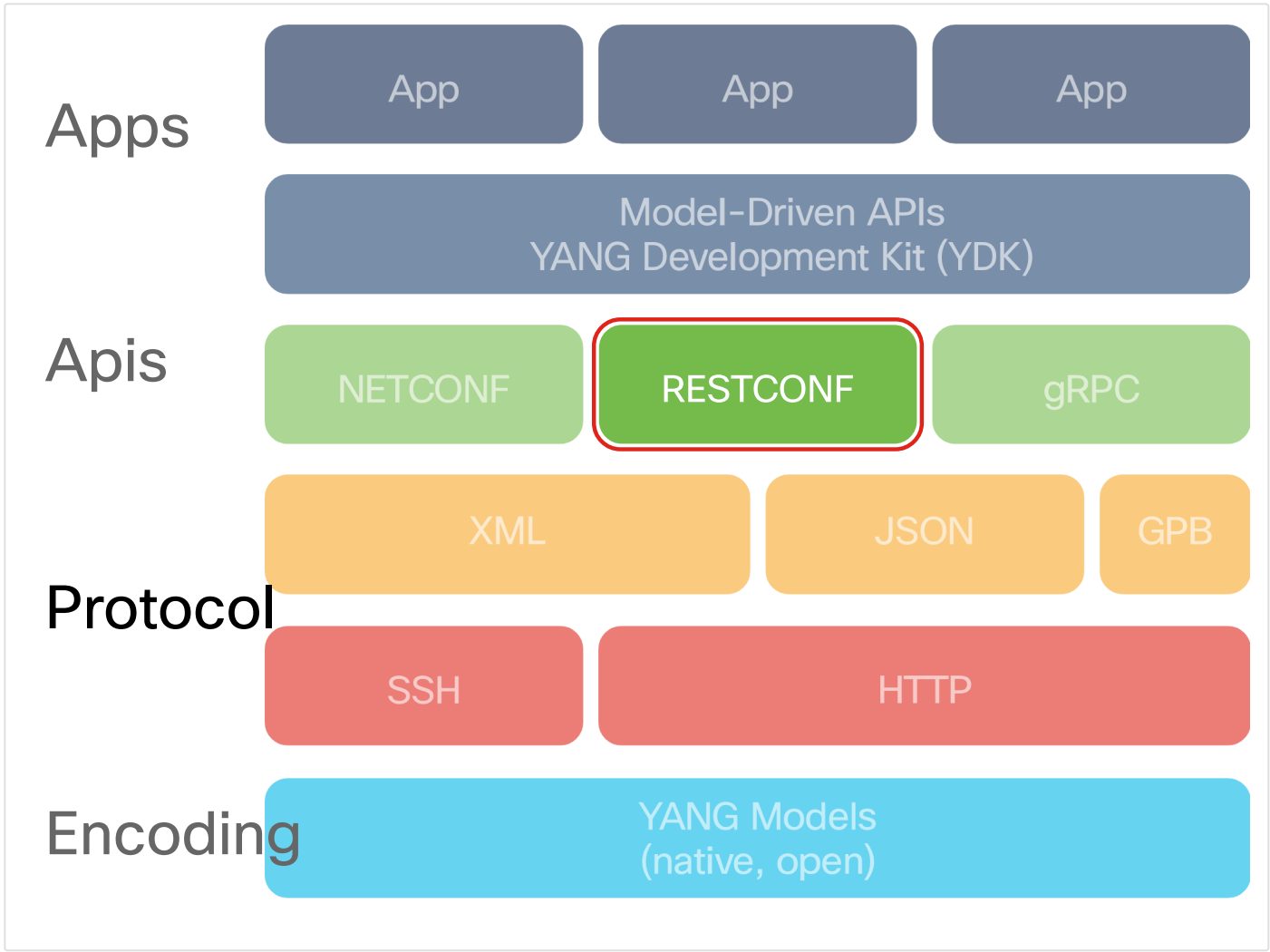
Note: You will learn how to use NETCONF in the lab, Use NETCONF to Access an IOS XE Device.

8.3.4

What is RESTCONF?

The RESTCONF RFC 8040 defines a protocol and mechanism for REST-like access to configuration information and control. Similar to NETCONF, it uses the datastore models and command verbs defined in the Network Configuration Protocol (NETCONF), encapsulated in HTTP messages. As with NETCONF, the YANG language is used to define the command structure syntax, as the semantics of the configuration datastore configuration, state, and events.

RESTCONF in the Model-Driven Programmability Stack



RESTCONF uses structured data (XML or JSON) and YANG to provide REST-like APIs, enabling programmatic access to devices. HTTP commands GET, POST, PUT, PATCH, and DELETE are directed to a RESTCONF API to access data resources represented by YANG data models. These basic edit operations allow the running configuration to be viewed and altered by a RESTCONF client.

RESTCONF is not intended to replace NETCONF, but rather to provide an HTTP interface that follows the REST principles and is compatible with the NETCONF datastore model. A network device may serve both NETCONF and RESTCONF concurrently, or may provide only one or the other.

Modeling

RESTCONF vs. NETCONF

Overall, NETCONF is more comprehensive, flexible, and complex than RESTCONF. RESTCONF is easier to learn and use for engineers with previous REST API experience. The following are differences between NETCONF and RESTCONF:

- NETCONF supports running and candidate data stores, while RESTCONF supports only a running datastore as any edits of candidate data store are immediately committed.
- RESTCONF does not support obtaining or releasing a data store lock. If a datastore has an active lock, the RESTCONF edit operation will fail.
- A RESTCONF edit is a transaction limited to a single RESTCONF call.
- RESTCONF does not support transactions across multiple devices.
- Validation is implicit in every RESTCONF editing operation, which either succeeds or fails.

NETCONF operations and RESTCONF methods mapping

Description	NETCONF	RESTCONF
Create a data resource	<edit-config>, </edit-config>	POST
Retrieve data and metadata	<get-config>, <get> , </get-config>	GET
Create or replace a data resource	<edit-config> (nc:operation="create/replace")	PUT
Delete a data resource	<edit-config> (nc:operation="delete")	DELETE

Each device RESTCONF server is accessed via API methods. Where can you find the RESTCONF API? The answer is that individual device APIs are rarely published. Instead the method URLs are determined dynamically.

The RESTCONF RFC 8040 states that RESTCONF base URI syntax is `/restconf/<resource-type>/<yang-module:resource>`. `<resource-type>` and `<yang-module:resource>` are variables and the values are obtained using specific YANG model files.

The basic format of a RESTCONF URL is `https://<hostURL>/restconf<resource><container><leaf><options>` where any portion after `restconf` could be omitted.

An initial GET request with just the `restconf` portion should return a response with the resources available at that server. For example, a GET constructed as follows:

```
GET /restconf HTTP/1.1
Host: example.com
Accept: application/yang-data+json
```

Would return something similar to:

```
HTTP/1.1 200 OK
Date: Thu, 26 Jan 2017 20:56:30 GMT
Server: example-server
Content-Type: application/yang-data+json
    {
        "ietf-restconf:restconf" : {
            "data" : {},
            "operations" : {},
            "yang-library-version" : "2016-06-21"
        }
    }
```

This response tells us that `data`, and `operations` are RESTCONF resource-types supported by this device.

Note: You will learn how to use RESTCONF in the lab, Use RESTCONF to Access an IOS XE Device.

8.3.5

Lab – Explore YANG Models

In this lab, you will learn how to use the open source pyang tool to transform YANG data models from files using the YANG language, into a much easier to read format. Using the “tree” view transformation, you will identify the key elements of the ietf-interfaces YANG model.

You will complete these objectives:

- Part 1: Launch the DEVASC VM and CSR1000v VMs
- Part 2: Explore a YANG Model on GitHub
- Part 3: Explore a YANG Model Using pyang

Explore YANG Models

8.3.6

Lab – Use NETCONF to Access an IOS XE Device

In this lab, you will use a NETCONF client, ncclient, which is a Python module for client-side scripting. You will use ncclient to verify NETCONF is configured, retrieve a device configuration, and modify a device configuration.

You will complete these objectives:

- Part 1: Build the Network and Verify Connectivity
- Part 2: Use a NETCONF Session to Gather Information
- Part 3: Use ncclient to Connect to NETCONF
- Part 4: Use ncclient to Retrieve the Configuration
- Part 5: Use ncclient to Configure a Device
- Part 6: Challenge: Modify the Program Used in This Lab

Use NETCONF to Access an IOS XE Device

8.3.7

Lab – Use RESTCONF to Access an IOS XE Device

In the first half of this lab, you will use the Postman program to construct and send API requests to the RESTCONF service that is running on R1. In the second half of the lab, you will create Python scripts to perform the same tasks as your Postman program.

You will complete these objectives:

- Part 1: Build the Network and Verify Connectivity
- Part 2: Configure an IOS XE Device for RESTCONF Access
- Part 3: Open and Configure Postman
- Part 4: Use Postman to Send GET Requests
- Part 5: Use Postman to Send a PUT Request
- Part 6: Use a Python Script to Send GET Requests
- Part 7: Use a Python Script to Send a PUT Request

Use RESTCONF to Access an IOS XE Device

8.2
Cisco SDKs

8.4
Cisco Network Management