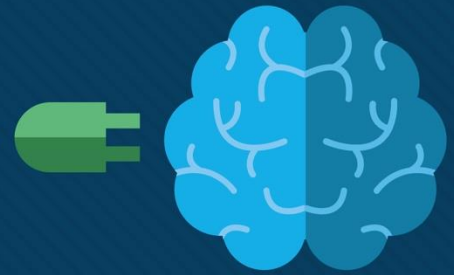

Chapter 28: Foundational Network Programmability Concepts

Instructor Materials

CCNP Enterprise: Core Networking

Chapter 28 Content

This chapter covers the following content:

Command-Line Interface (CLI) - This section provides an overview of the pros and cons of managing devices with the traditional command-line interface approach.

Application Programming Interface (API) - This section describes what APIs are, the different types of APIs, and how they are used.

Data Models and Supporting Protocols - This section describes some of the most common data models and associated tools

Cisco DevNet - This section provides a high-level overview of the various Cisco DevNet components and learning labs.

Chapter 28 Content (Cont.)

GitHub - This section illustrates different use cases for version control and the power of community code sharing.

Basic Python Components and Scripts - This section illustrates the components of Python scripts and how to interpret them.

Command-Line Interface (CLI)

The biggest flaw with using the CLI to manage a network is misconfiguration.

Command-Line Interface (CLI)

The most glaring and biggest flaws with using the CLI to manage a network is misconfiguration.

When businesses have increased complexity in their networks, the cost of something failing can be very high due to the increased time it takes to troubleshoot the issues in a complex network.

There are tools that can assist in reducing the number of outages that are caused by human error due to misconfigurations in the CLI.

Table 28-2 CLI PROs and CONs

PROs	CONs
Well known and documented	Difficult to scale
Commonly used method	Large number of commands
Commands can be scripted	Must know IOS command syntax
Syntax help available on each command	Executing commands can be slow
Connection to CLI can be encrypted (using SSH)	Not intuitive
	Can execute only one command at time
	CLI and commands can change between software versions and platforms
	Using the CLI can pose a security threat if using Telnet (plaintext)

Application Programming Interface

- Northbound and Southbound APIs
- Representational State Transfer (REST) APIs
- Introduction to Postman
- Data Formats (XML and JSON)
- Cisco DNA Center APIs
- Cisco vManage APIs

Northbound and Southbound APIs

A method of communicating with and configuring a network is the use of application programming interfaces (APIs).

The two most common APIs: the Northbound and Southbound APIs which are used in network automation.

Northbound APIs are often used to communicate from a network controller to its management software. Cisco DNA Center has a software GUI that is used to manage the network controller.

The controller information that is being passed from the management software is leveraging a Northbound REST-based API. This traffic should be encrypted using TLS.

If a change to a switch's configuration in the management software of the controller, those changes are then pushed down to the individual devices like routers, switches and WAPs using a Southbound API and programmatic interface.

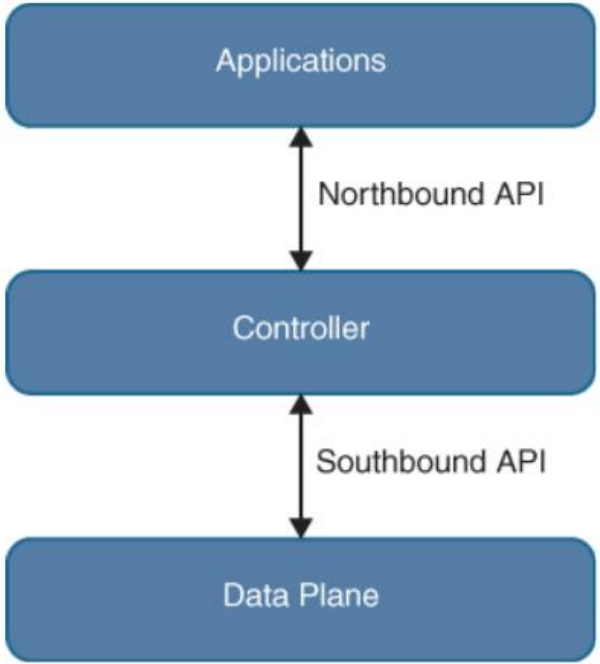

Figure 28-1 *Basic API Operations*

Representational State Transfer (REST) APIs

An API that uses REST is often referred to a RESTful API. RESTful APIs use HTTP methods to gather and manipulate data. HTTP offers a consistent way to interact with APIs from multiple vendors.

HTTP functions are similar to functions that most applications or databases use to store or alter data. These functions are called "CRUD" functions: CREATE, READ, UPDATE, and DELETE.

HTTP Function	Action	Use Case
GET	Requests data from a description	Viewing a website
PPOST	Submits data to a specific destination	Submitting login credentials
PUT	Replaces data in a specific destination	Updating an NTP server
PATCH	Appends data to a specific destination	Adding an NTP server
DELETE	Removes data from a specific destination	Removing an NTP server

Table 28-3 HTTP Functions and Use Cases

CRUD Function	Action	Use Case
CREATE	Inserts data in a database or application	Updating a customer's home address in a database
READ	Retrieves data from a database or application	Pulling up a customer's home address from a database
UPDATE	Modifies or replaces data in a database or application	Changing a street address stored in a database
DELETE	Removes data from a database or application	Removing a customer from a database

Table 28-4 CRUD Functions and Use Cases

Introduction to Postman

One of the most important pieces of interacting with any software using APIs is testing. Testing code helps ensure that developers are accomplishing the outcome that was intended when executing the code.

Many APIs require user authentication to gain access to utilize the APIs.

If a REST API call is used to delete data, that data will be removed just as if a user logged in via the CLI and deleted it.

Postman is an application that makes it possible to interact with APIs using a console-based approach. Postman allows for the use of various data types and formats to interact with REST-based APIs. Figure 28-2 shows the main Postman application dashboard.

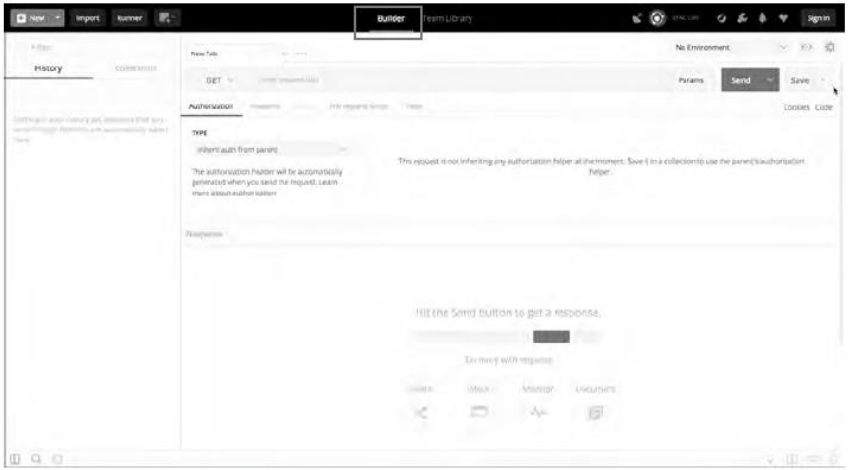

Figure 28-2 *Postman Dashboard*

Introduction to Postman

One of the most important pieces of interacting with any software using APIs is testing. Testing code helps ensure that developers are accomplishing the outcome that was intended when executing the code.

Many APIs require user authentication to gain access to utilize the APIs.

If a REST API call is used to delete data, that data will be removed just as if a user logged in via the CLI and deleted it.

Postman is an application that makes it possible to interact with APIs using a console-based approach. Postman allows for the use of various data types and formats to interact with REST-based APIs.

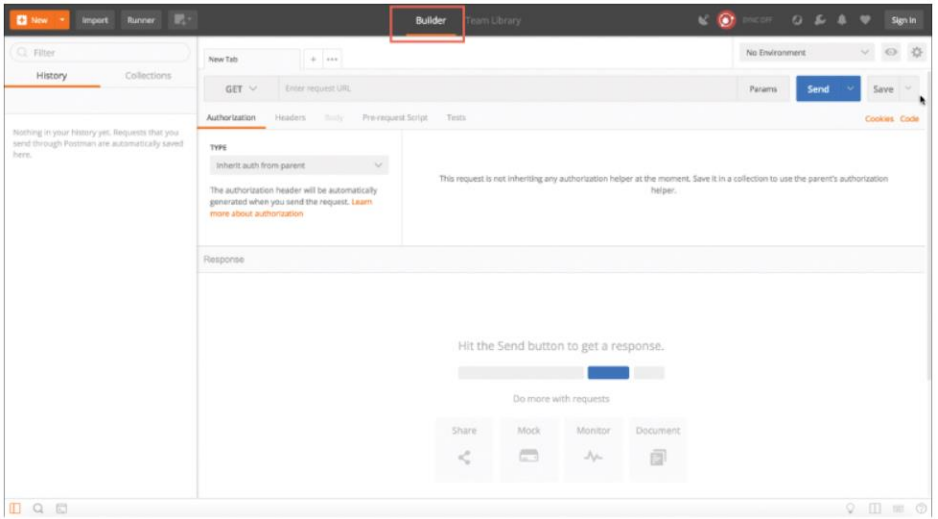

Figure 28-2 *Postman Dashboard*

Introduction to Postman (Cont.)

The Postman application has various sections that you can interact with. The focus here is on using the Builder portion of the dashboard.

- **History** - The history tab shows a list of all the recent API calls made.
- **Collections** - API calls can be stored in groups, called collections, that are specific to a user's needs.
- **New Tab** - Tabs provide another very convenient way to work with various API calls. Each tab can have its own API call and parameters that are completely independent of any other tab.
- **URL Bar** - Each tab has its own URL bar to be able to use a specific API. Remember that an API call using REST is very much like an HTTP transaction. Each API call in a RESTful API maps to an individual URL for a particular function. This means every configuration change or poll to retrieve data a user makes in a REST API has a unique URL—whether it is a GET, POST, PUT, PATCH, or DELETE function.

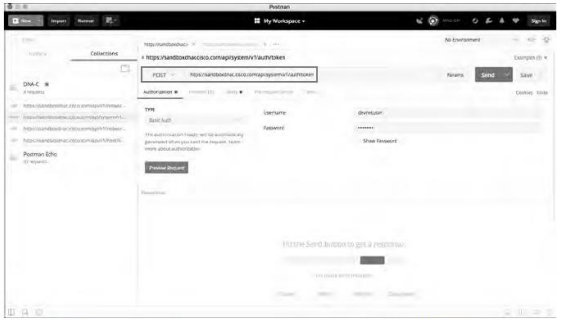

Figure 28-5 *Postman URL Bar with Cisco DNA Center Token API Call*

Data Formats XML and JSON

XML and JSON are two of the most common data formats that are used with APIs. Extensible Markup Language (XML) is commonly used when constructing web services.

XML is a tag-based language. For example, a start tag named interface is represented as <interface> and the end tag for <interface> would be </interface>.

Inside the start tag and end tag, you can use different code and parameters. Example 28-1 shows a snippet of XML output with both start and end tags as well as some configuration parameters.

Notice that each section of Example 28-1 has a start tag and an end tag. The data is structured so that it contains a section called “users,” and within that section are four individual users:

- root
- Jason
- Jamie
- Luke

Example 28-1 *XML Code Snippet*

```
<users>
   <user>
     <name>root</name>
   </user>
   <user>
     <name>Jason</name>
   </user>
   <user>
     <name>Jamie</name>
   </user>
   <user>
     <name>Luke</name>
   </user>
 </users>
```

Data Formats XML and JSON (Cont.)

JavaScript Object Notation (JSON) is much easier to work with than XML. It is simple to read and create.

JSON stores all its information in key/value pairs.

JSON uses objects for its format. Each JSON object starts with a { and ends with a }.

Example 28-3 shows how JSON can be used to represent the same username example shown for XML in Example 28-1. You can see that it has four separate key/value pairs, one for each user's name.

In this JSON code snippet, you can see that the first key is user, and the value for that key is a unique username, root.

Example 28-3 *JSON Code Snippet*

```
{
  "user": "root",
  "father": "Jason",
  "mother": "Jamie",
  "friend": "Luke"
}
```

HTTP Status Codes

Now that the XML and JSON data formats have been explained, it is important to circle back to using the REST API and the associated responses and outcomes of doing so.

First, we need to look at the HTTP response status codes. Most internet users have experienced the dreaded “404 Not Found” error when navigating to a website.

Table 28-5 lists the most common HTTP status codes as well as the reasons users may receive each one.

HTTP Status Code	Result	Common Reason for Response Code
200	OK	Using GET or POST to exchange data with an API
201	Created	Creating resources by using a REST API call
400	Bad Request	Request failed due to client-side issue
401	Unauthorized	Client not authenticated to access site or API call
403	Forbidden	Access not granted based on supplied credentials
404	Not Found	Page at HTTP URL location does not exist or is hidden

Cisco DNA Center APIs

The Cisco DNA Center controller expects all incoming data from the REST API to be in JSON format. The HTTP POST function is used to send the credentials to the Cisco DNA Center controller.

Cisco DNA Center uses basic authentication to pass a username and password to the Cisco DNA Center Token API to authenticate users. This API is used to authenticate a user to the Cisco DNA Center controller to make additional API calls.

The key steps necessary to successfully set up the API call in Postman are as follows:

Step 1. In the URL bar, enter https://sandboxdnac.cisco.com/api/system/v1/auth/token to target the Token API.
Step 2. Select the HTTP POST operation from the dropdown box.
Step 3. Under the Authorization tab, ensure that the type is set to Basic Auth.
Step 4. Enter devnetuser as the username and Cisco123! as the password.
Step 5. Select the Headers tab and enter Content-Type as the key.
Step 6. Select application/json as the value.
Step 7. Click the Send button to pass the credentials to the Cisco DNA Center controller via the Token API.

Cisco DNA Center APIs (Cont.)

You can tell that the API call to the Cisco DNA Center controller completed successfully by receiving an HTTP status code 200.

The Network Device API allows users to retrieve a list of devices that are currently in inventory that are being managed by the Cisco DNA Center controller.

You need to prepare Postman to use the token that was generated when you successfully authenticated:

Step 1. Copy the token you received earlier and click a new tab in Postman.
Step 2. In the URL bar enter https://sandboxdnac.cisco.com/api/v1/network-device
Step 3. Select the HTTP GET operation from the dropdown box.
Step 4. Select the Headers tab and enter Content-Type as the key.
Step 5. Select application/json as the value.
Step 6. Add another key and enter X-Auth-Token.
Step 7. Paste the token in as the value.
Step 8. Click Send to pass the token to the Cisco DNA Center controller and perform an HTTP GET to retrieve a device inventory list.

Cisco DNA Center APIs (Cont.)

Within a few moments an API call can be used to gather that data for the entire network.

When using APIs, it is common to manipulate data by using filters and offsets. If a user wants to leverage the Network Device API to gather information on only the second device in the inventory. This is where the API documentation becomes so valuable. Most APIs have documentation that explains what they can be used to accomplish.

In Postman, it is possible to modify the Network Device API URL and add ?limit=1 to the end of the URL to show only a single device in the inventory. It is also possible to add the &offset=2 command to the end of the URL to state that only the second device in the inventory should be shown. These query parameters are part of the API.

Example 28-4 *Device Inventory Pulled Using a Network Device API Call in Postman*


```
{
    "response": [
        {
            "type": "Cisco ASR 1001-X Router",
            "family": "Routers",
            "location": null,
            "errorCode": null,
            "macAddress": "00:c8:8b:80:bb:00",
            "lastUpdateTime": 1521645053028,
            "apManagerInterfaceIp": "",
            "associatedWlcIp": "",
            "bootDateTime": "2018-01-11 15:47:04",
            "collectionStatus": "Managed",
            "interfaceCount": "10",
            "lineCardCount": "9",
            "lineCardId": "a2406c7a-d92a-4fe6-b3d5-ec6475be8477,
5b75b5fd-21e3-4deb-a8f6-6094ff73e2c8, 8768c6f1-e19b-4c62-a4be-51c001b05b0f,
afdfa337-bd9c-4eb0-ae41-b7a97f5f473d, c59fbb81-d3b4-4b5a-81f9-fe2c8d80aead,
b21b6024-5dc0-4f22-bc23-90fc618552e2, 1be624f0-1647-4309-8662-a0f87260992a,
56f4fbb8-ff2d-416b-a7b4-4079acc6fa8e, 164716c3-62d1-4e48-a1b8-42541ae6199b",
            "managementIpAddress": "10.10.22.74",
            "memorySize": "3956371104",
            "platformId": "ASR1001-X",
            "reachabilityFailureReason": "",
            "reachabilityStatus": "Reachable",
            "series": "Cisco ASR 1000 Series Aggregation Services Routers",
            "snmpContact": "",
            "snmpLocation": "",
            "tunnelUdpPort": null,
            "waasDeviceMode": null,
            "locationName": null,
            "role": "BORDER ROUTER",
            "hostname": "asr1001-x.abc.inc",
            "upTime": "68 days, 23:23:31.43",
            "inventoryStatusDetail": "<status><general code=\"SUCCESS\"/></status>",
            "softwareVersion": "16.6.1",
            "roleSource": "AUTO",
            "softwareType": "IOS-XE",
            "collectionInterval": "Global Default",
            "lastUpdated": "2018-03-21 15:10:53",
            "tagCount": "0",
            "errorDescription": null,
            "serialNumber": "FXS1932Q1SE",
            "instanceUuid": "d5bbb4a9-a14d-4347-9546-89286e9f30d4",
            "id": "d5bbb4a9-a14d-4347-9546-89286e9f30d4"
        },
Output Snipped for brevity
```

Cisco vManage APIs

There are various APIs available in the Cisco SD-WAN and the vManage controller.

With a Cisco SD-WAN API you need to provide login credentials to the API in order to be able to utilize the API calls.

Some key pieces of information are necessary to successfully set up the API call in Postman:

- The URL bar must have the API call to target the Authentication API
- The HTTP POST operation is used to send the username and password to Cisco vManage
- The Headers Content-Type key must be application/x-www-form-urlencoded
- The body must contain keys with the j_username devnetuser and thej_password Cisco123!

Data Models and Supporting Protocols

This section provides a high-level overview of some of the most common data models and tools and how they are leveraged in a programmatic approach:

- Yet Another Next Generation (YANG) modeling language
- Network Configuration Protocol (NETCONF)
- RESTCONF

YANG Data Models

YANG uses data models. Data models are used to describe whatever can be configured, monitored or executed on a device.

Data models create a uniform way to describe data across vendor platforms.

YANG models use a tree structure. The tree structure represents how to reach a specific element of the model, and the elements can be either configurable or not configurable. Every element has a defined type. For example, an interface can be configured to be on or off.

Example 28-5 can be read as follows: There is food. Of that food, there is a choice of snacks. The snack choices are pretzels and popcorn. If it is late at night, the snack choices are two different types of chocolate. A choice must be made to have milk chocolate or dark chocolate, and if the consumer is in a hurry and does not want to wait, the consumer can have the first available chocolate, whether it is milk chocolate or dark chocolate.

Example 28-5 *YANG Model Example*

```
container food {
  choice snack {
      case sports-arena {
          leaf pretzel {
              type empty;
          }
          leaf popcorn {
              type empty;
          }
      }
      case late-night {
          leaf chocolate {
              type enumeration {
                  enum dark;
                  enum milk;
                  enum first-available;
              }
          }
      }
  }
}
```

YANG Data Models (Cont.)

The YANG model in Example 28-6 can be read as follows:

There is a list of interfaces. Of the available interfaces, there is a specific interface that has three configurable speeds.

Those speeds are 10 Mbps, 100 Mbps, and auto, as listed in the leaf named speed.

The leaf named observed-speed cannot be configured due to the config false command. This is because as the leaf is named, the speeds in this leaf are what was auto-detected (observed); hence, it is not a configurable leaf. This is because it represents the auto-detected value on the interface, not a configurable value.

Example 28-6 *Network-Oriented YANG Model*

```
list interface {
     key "name";

     leaf name {
         type string;
     }
     leaf speed {
         type enumeration {
             enum 10m;
             enum 100m;
             enum auto;
         }
     }
     leaf observed-speed {
         type uint32;
         config false;
     }
 }
```

NETCONF

NETCONF is an IETF standard protocol that uses the YANG data models to communicate with the various devices on the network. NETCONF runs over SSH, TLS, and (although not common) SOAP.

Some of the key differences between SNMP and NETCONF are listed in Table 28-6.

One of the most important differences is that SNMP can't distinguish between configuration data and operational data, but NETCONF can. Another key differentiator is that NETCONF uses paths to describe resources, whereas SNMP uses object identifiers (OIDs).

A NETCONF path can be similar to interfaces/interface/eth0, which is much more descriptive than what you would expect from SNMP.

- Collecting the status of specific fields
- Changing the configuration of specific fields
- Taking administrative actions
- Sending event notifications
- Backing up and restoring configurations
- Testing configurations before finalizing the transaction

Feature	SNMP	NETCONF
Resources	OIDs	Paths
Data Models	Defined in MIBs	YANG core models
Data modeling language	SMI	YANG
Management operations	SNMP	NETCONF
Encoding	BER	XML,JSON
Transport stack	UDP	SSH/TCP

NETCONF (Cont.)

Figure 28-15 illustrates how NETCONF uses YANG data models to interact with network devices and then talk back to management applications. The dotted lines show the devices talking back directly to the management applications, and the solid lines illustrate the NETCONF protocol talking between the management applications and the devices.

NETCONF exchanges information called capabilities when the TCP connection has been made. Capabilities tell the client what the device it's connected to can do. Furthermore, other information can be gathered by using the common NETCONF operations shown in Table 28-7.

Information and configurations are stored in datastores. Datastores can be manipulated by using the NETCONF operations listing in Table 28-7. NETCONF uses Remote Procedure Call (RPC) messages in XML format to send the information between hosts.

NETCONF Operation	Description
<get>	Requests running configuration and state information of the device
<get-config>	Requests some or all of the configuration from a datastore
<edit-config>	Edits a configuration datastore by using CRUD operations
<copy-config>	Copies the configuration to another datastore
<delete-config>	Deletes the configuration

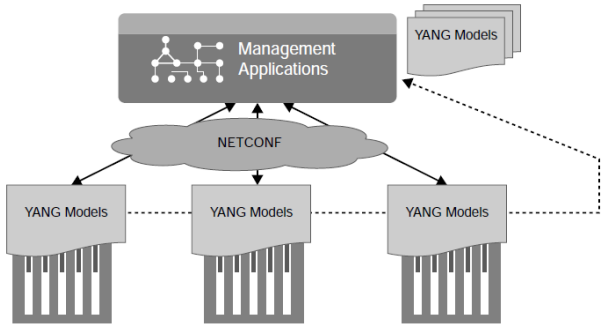

Figure 28-15 *NETCONF/YANG Interfacing with Management Applications*

NETCONF (Cont.)

Example 28-9 shows an example of an OSPF NETCONF RPC message that provides the OSPF routing configuration of an IOS XE device.

The same OSPF router configuration that would be seen in the command-line interface of a Cisco router can be seen using NETCONF.

The data is just structured in XML format rather than what users are accustomed to seeing in the CLI. It is easy to read the output in these examples because of how legible XML is.

Example 28-10 *NETCONF Save Config Example*

```
<?xml version="1.0" encoding="utf-8"?>
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="">
  <cisco-ia:save-config xmlns:cisco-ia="http://cisco.com/yang/cisco-ia"/>
</rpc>
```

Example 28-9 *NETCONF OSPF Configuration Example*

```
<rpc-reply message-id="urn:uuid:0e2c04cf-9119-4e6a-8c05-238ee7f25208"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" xmlns:nc="urn:ietf:params:
xml:ns:netconf:base:1.0">
  <data>
    <native xmlns="http://cisco.com/ns/yang/ned/ios">
      <router>
        <ospf>
          <id>100</id>
          <redistribute>
            <connected>
              <redist-options>
                <subnets/>
              </redist-options>
            </connected>
          </redistribute>
          <network>
            <ip>10.10.0.0</ip>
            <mask>0.0.255.255</mask>
            <area>0</area>
          </network>
          <network>
            <ip>20.20.0.0</ip>
            <mask>0.0.255.255</mask>
            <area>0</area>
          </network>
          <network>
            <ip>100.100.0.0</ip>
            <mask>0.0.255.255</mask>
            <area>0</area>
          </network>
        </ospf>
      </router>
    </native>
  </data>
</rpc-reply>
```

RESTCONF

RESTCONF is used to programmatically interface with data defined in YANG models while also using the datastore concepts defined in NETCONF.

RESTCONF uses the same YANG models as NETCONF and Cisco IOS XE. The goal of RESTCONF is to provide a RESTful API experience while still leveraging the device abstraction capabilities provided by NETCONF. RESTCONF supports the following HTTP methods and CRUD operations:

GET, POST, PUT, DELETE, OPTIONS

The RESTCONF requests and responses can use either JSON or XML structured data formats.

Example 28-11 shows a brief example of a RESTCONF GET request on a Cisco router to retrieve the logging severity level that is configured. This example uses JSON instead of XML. Notice the HTTP status 200, which indicates that the request was successful.

Example 28-11 *RESTCONF GET Logging Severity Example*

```
RESTCONF GET

------------------------

URL: https://10.85.116.59:443/restconf/data/Cisco-IOS-XE-native:native/logging/
monitor/severity

Headers: {'Accept-Encoding': 'gzip, deflate',  'Accept': 'application/
yang-data+json, application/yang-data.errors+json'}

Body:

RESTCONF RESPONSE

---------------------------

200

{

  "Cisco-IOS-XE-native:severity": "critical"

}
```

Cisco DevNet

- Network operators who are looking to enhance or increase their skills with APIs, coding, Python, or even controller concepts can find a wealth of help at DevNet.
- At DevNet it is easy to find learning labs and content to help solidify current knowledge in network programmability.

DEVNET API Labs

This section provides a high-level overview of DevNet, including the different sections of DevNet and some of the labs and content that are available.

Figure 28-16 shows the DevNet main page.

The Discover page is where you can navigate the different offerings that DevNet has available.

Under this tab are subsections for guided learning tracks, which guide you through various technologies and the associated API labs.

Some of the labs you interact with are: Programming the Cisco Digital Network Architecture (DNA), ACI Programmability, Getting Started with Cisco WebEx Teams APIs, and Introduction to DevNet.

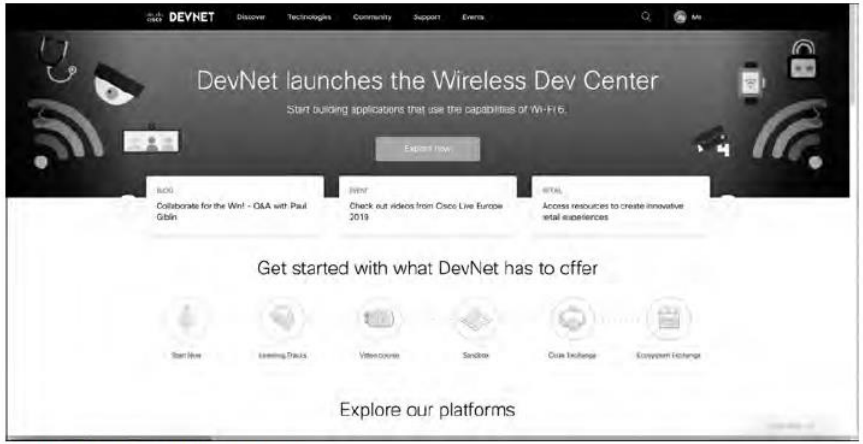

Figure 28-16 *DevNet Main Page*

- **Discover**
- **Technologies**
- **Community**
- **Support**
- **Events**

DEVNET API Labs (Cont.)

The Technologies page allows you to pick relevant content based on the technology you want to study.

The Community page is where users have access to many different people at various stages of learning. This is also the place to read blogs, sign up for developer forums, and follow DevNet on all major social media platforms.

The Support section of DevNet is where users can post questions and get answers from some of the best in the industry. You can ask questions about specific labs or the overarching technology (for example, Python or YANG models).

The DevNet Events page provides a list of all events that have happened in the past and that will be happening in the future. This is where a user can find the upcoming DevNet Express events as well as conferences.

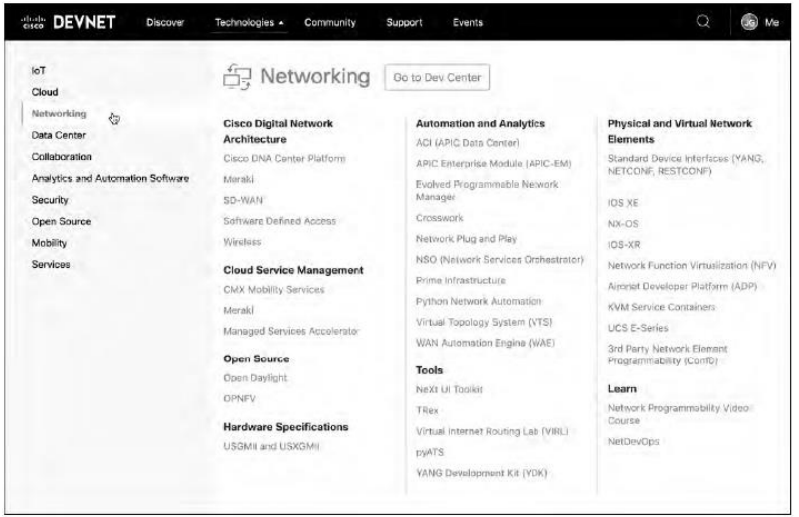

Figure 28-17 *DevNet Technologies Page*

GitHub

- One of the most efficient and commonly adopted ways of using version control.
- The ability to share your code and collaborate with an online community of programmers and developers.

GitHub Repositories and Version Control

One of the most commonly adopted ways of using version control is by using GitHub.

GitHub is a hosted web-based repository for code.

Using GitHub offers: easiest ways to track changes in your files, collaborate with other developers, and share code with the online community.

GitHub provides a guide that steps through how to create a repository, start a branch, add comments, and open a pull request.

Projects are repositories that contain code files. GitHub provides a single pane to create, edit, and share code files.

Figure 28-21 shows a repository called ENCORE that contains three files: ENCORE.txt, JSON_Example.txt, README.md

Figure 28-21 *GitHub ENCORE Repository*

GitHub Repositories and Version Control (Cont.)

GitHub gives a summary of commit logs, so when you save a change in one of your files or create a new file, GitHub shows details about it on the main repository page.

In the JSON_Example.txt, for example, GitHub shows its contents and how to edit the file in the repository.

Figure 28-22 shows the contents of the JSON_Example.txt file and the options available with the file.

This editor is very similar to any text editor.

Other GitHub users and developers can contribute to this code or add and delete lines of code based on the code that was originally created.

Figure 28-22 *JSON_Example.txt Contents*

Basic Python Components and Scripts

- Python has become one of the most common programming languages in terms of network programmability.
- This section leverages the new knowledge you have gained in this chapter about APIs, HTTP operations, DevNet, and GitHub.

Interpreting Python Scripts

When you understand the basics of interpreting what a Python script is designed to do, it will be easier to understand and leverage other scripts that are available.

Example 28-12 shows a Python script that sets up the environment to log in to the Cisco DNA Center sandbox. It uses the same credentials used with the Token API earlier.

Sandbox - The line that says ENVIRONMENT IN USE="sandbox" corresponds to the selection of the sandbox type of lab environments available through DevNet.

Express - This is the back end that is used for the DevNet Express Events that are held globally at various locations and Cisco office locations.

Custom - This is used in the event that there is already a Cisco DNA Center installed either in a lab or another facility, and it needs to be accessed using this script.

Example 28-12 *Env_Lab.py*

```
"""Set the Environment Information Needed to Access Your Lab!

The provided sample code in this repository will reference this file to get the
information needed to connect to your lab backend.  You provide this info here
once and the scripts in this repository will access it as needed by the lab.


Copyright (c) 2018 Cisco and/or its affiliates.
# User Input

# Please select the lab environment that you will be using today
#     sandbox - Cisco DevNet Always-On / Reserved Sandboxes
#     express - Cisco DevNet Express Lab Backend
#     custom  - Your Own "Custom" Lab Backend
ENVIRONMENT_IN_USE = "sandbox"

# Set the 'Environment Variables' based on the lab environment in use
if ENVIRONMENT_IN_USE == "sandbox":
    dnac = {
        "host": "sandboxdnac.cisco.com",
        "port": 443,
        "username": "devnetuser",
        "password": "Cisco123!"
    }
```

Interpreting Python Scripts (Cont.)

Variables are used to target the DevNet Cisco DNA Center sandbox specifically. Table 28-8 describes these variables.

The variables shown are in the JSON data format that was discussed earlier in this chapter. Remember that JSON uses key/value pairs and is extremely easy to read and interpret.

In Example 28-13, you can see the key/value pair “username”: “devnetuser”. The structure used to hold all the key/value pairs in this script is called a dictionary. In this particular Python script, the dictionary is named dnac. The dictionary named dnac contains multiple key/value pairs, and it starts and ends with curly braces ({})

Variable	Value	Description
host	sandboxdnac.cisco.com	Cisco DNA Center sandbox URL
port	443	TCP port to access URL securely (HTTPS)
username	devnetuser	Username to log in to Cisco DNA Center sandbox (via API or GUI)
password	Cisco123!	Password to log in to Cisco DNA Center sandbox (via API or GUI)

Table 28-8 Python Variables for Cisco DNA Center Sandbox in Env_Lab.py

Example 28-13 *Dictionary Used in Env_Lab.py*

```
dnac = {
    "host": "sandboxdnac.cisco.com",
    "port": 443,
    "username": "devnetuser",
    "password": "Cisco123!"
}
```

Interpreting Python Scripts (Cont.)

In the get_dnac_device.py script.

The first section of code tells the Python interpreter what modules this particular script will use.

Think of a module as a collection of actions and instructions. To better explain the contents in this script, comments are inserted throughout the script to help document each section.

Example 28-17 shows the first section of the get_dnac_devices.py with comments.

Example 28-17 *Explanation of the First Section of get_dnac_devices.py*

```
# Specifies which version of Python will be used
#! /usr/bin/env python3

# Calls "dnac" dictionary from the env_lab.py script covered earlier
from env_lab import dnac

# Imports JSON module so Python can understand the data format that contains key/
value pairs
import json

# Imports requests module which handles HTTP headers and form data
import requests

# Imports urllib3 module which is an HTTP client
import urllib3

# Imports HTTPBasicAuth method from the requests.auth module for authentication to
Cisco DNA Center
from requests.auth import HTTPBasicAuth

# Imports prettytable components from PrettyTable module to structure return data
from Cisco DNA Center in table format
from prettytable import PrettyTable
```

Interpreting Python Scripts (Cont.)

Modules help Python understand what it is capable of.

If a developer tried to do an HTTP GET request without having the Requests modules imported, it would be difficult for Python to understand how to interpret the HTTP call. Although there are other ways of doing HTTP calls from Python, the Requests modules greatly simplify this process.

Example 28-18 shows the second section of the get_dnac_devices.py script.

Example 28-18 *Explanation of the Second Section of get_dnac_devices.py*

```
# Puts return data from Cisco DNA Center Network Device API call into easily read-
able table with column names Hostname, Platform Id, Software Type, Software Version
and Up Time.
dnac_devices = PrettyTable(['Hostname','Platform Id','Software Type','Software
Version','Up Time' ])
dnac_devices.padding_width = 1

# Silences the insecure warning due to SSL Certificate
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)

# Sends specific HTTP headers to Cisco DNA Center when issuing HTTP GET to the Net-
work Devices API

headers = {
              'content-type': "application/json",
              'x-auth-token': ""
          }
```

Interpreting Python Scripts (Cont.)

Functions are blocks of code that are built to perform specific actions. Functions are very structured in nature and can often be reused later on within a Python script.

Some functions are built into Python and do not have to be created. The print function, which can be used to print data to a terminal screen. You can see the print function at the end of the get_dnac_devices.py script.

In order to execute any API calls to Cisco DNA Center, you must be authenticated, using the Token API.

Example 28-19 shows the use of the Token API within a Python script. (Recall that you saw this API used with Postman earlier in the chapter.)

Example 28-19 *Explanation of the Third Section of get_dnac_devices.py*

```
# This function does an HTTP POST of the username devnetuser and the password of
Cisco123! to the Token API located at https://sandboxdnac.cisco.com/api/system/v1/
auth/token and uses the values built in the JSON key-value pairs from the Env_Lab.
py. The JSON response from the API called is stored as the Token that will be used
for future API calls for this authenticated user.


def dnac_login(host, username, password):
    url = "https://{}/api/system/v1/auth/token".format(host)
    response = requests.request("POST", url, auth=HTTPBasicAuth(username, password),
                                headers=headers, verify=False)
    return response.json()["token"]
```

Interpreting Python Scripts (Cont.)

The script shown in Example 28-20 ties the Token API to the Network Device API call to retrieve the information from Cisco DNA Center.

The line that says header [“x-auth-token”] = token is mapping the JSON response from the previous example, which is the token, into the header called x-auth-token. In addition, the URL for the API has changed to network_device, and the response is sending a requests.get to that URL.

This is exactly the same example used with Postman earlier in this chapter.

Example 28-20 *Explanation of the Fourth Section of get_dnac_devices.py*

```
def network_device_list(dnac, token):
    url = "https://{}/api/v1/network-device".format(dnac['host'])
    headers["x-auth-token"] = token
    response = requests.get(url, headers=headers, verify=False)
    data = response.json()
    for item in data['response']:
        dnac_devices.add_row([item["hostname"],item["platformId"],item["softwareType
"],item["softwareVersion"],item["upTime"]])
```

Interpreting Python Scripts (Cont.)

The final section of get_dnac_devices.py shows code that ties the dnac dictionary that is in the Env_Lab.py script to the dnac_login function covered earlier. In addition, the print function takes the response received from the response.get that was sent to the Network Device API and puts it into the table format that was specified earlier in the script with the name dnac_devices.

Example 28-21 shows the final lines of code in the script.

Example 28-21 *Explanation of the Fifth Section of get_dnac_devices.py*

```
login = dnac_login(dnac["host"], dnac["username"], dnac["password"])
network_device_list(dnac, login)


print(dnac_devices)
```

Prepare for the Exam

Prepare for the Exam

Key Topics for Chapter 28

Description
HTTP Functions and Use Cases
CRUD Functions and Use Cases
HTTP Status Codes
Steps to authenticate to Cisco DNA Center using a POST operation and basic authentication
Steps to leverage the Network Device API to retrieve a device inventory from Cisco DNA Center
Using the offset and limit filters with the Network Device API when gathering device inventory

Prepare for the Exam

Key Terms for Chapter 28

Key Terms
Application Programming Interface (API)
command-line interface (CLI)
DevNet
Extensible Markup Language (XML)
GitHub
Java-Script Object (JSON)
NETCONF
Python
RESTCONF
Yang Model


CISCO