

TOPICS COVERED:

NETWORK PROGRAMMABILITY

- Application Programming Interface
- POSTMAN
- Cisco DNA API
- Cisco vManage API
- DATA FORMATS: XML, JSON
- DATA MODELLINGS & PROTOCOLS: YANG, NETCONF and RESTCONF
- CISCO IOS EMBEDDED EVENT MANAGER (EEM)
- PYTHON
- ANSIBLE
- NETWORK AUTOMATION SCRIPTS

Application Programming Interface

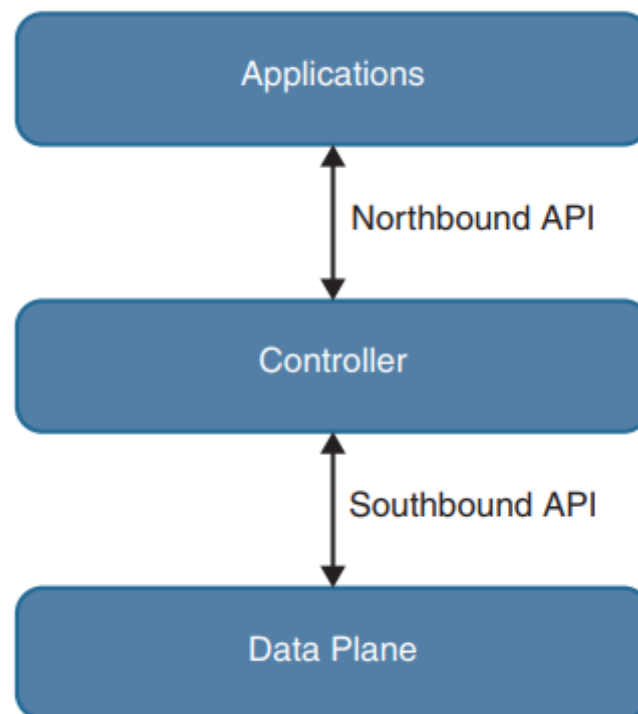
Another very popular method of communicating with and configuring a network is through the use of **application programming interfaces (APIs)**.

APIs are mechanisms used to communicate with **applications and other software**.

They are also **used to communicate with various components of the network through software**.

It is possible to use **APIs to configure or monitor specific components of a network**.

There are multiple different types of APIs. However, the focus is on two of the most common APIs: the **Northbound and Southbound APIs**.

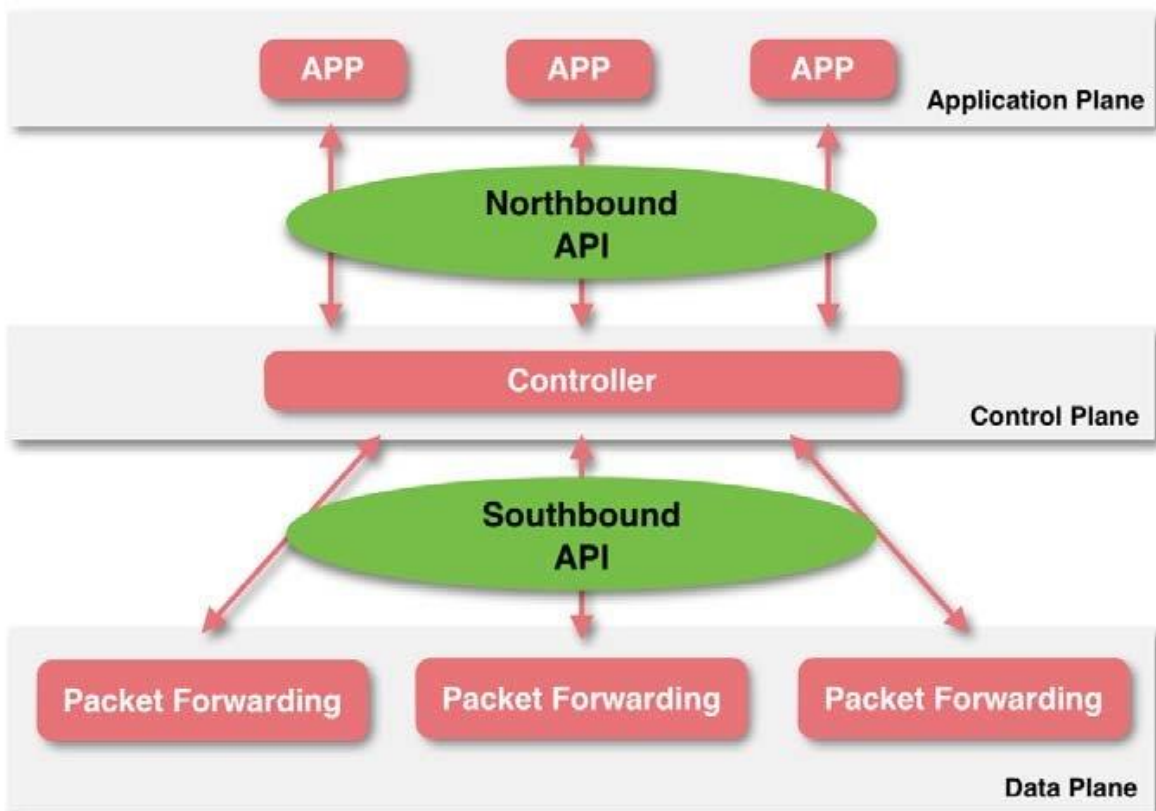
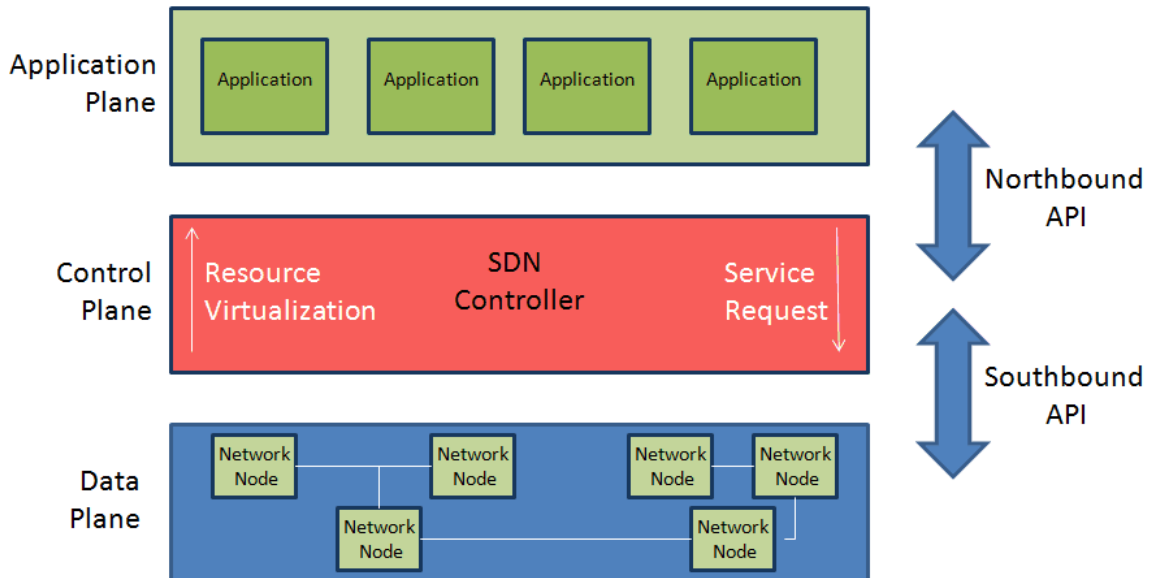


Northbound API

Northbound APIs are often used to communicate from a network controller to its management software. For example, **Cisco DNA Center** has a software graphical user interface (GUI) that is used to manage the **network controller**. Typically, when a network operator logs into a controller to manage the network, the information that is being passed from the management software is leveraging a **Northbound REST-based API**. Best practices suggest that the traffic should be **encrypted using TLS** between the software and the controller. Most types of APIs have the ability to use encryption to secure the data in flight.

Southbound API

If a network operator makes a change to a switch’s configuration in the management software of the controller, those changes are then pushed down to the individual devices by using a Southbound API. These devices can be routers, switches, or even wireless access points. APIs interact with the components of a network through the use of a programmatic interface.



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. Because there is a defined structure for how HTTP works, it offers a consistent way to interact with APIs from multiple vendors. REST uses different HTTP functions to interact with the data.

HTTP Function	Action	Use Case
GET	Requests data from a destination	Viewing a website
POST	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

HTTP functions are similar to the functions that most applications or databases use to store or alter data—whether the data is stored in a database or within the application. These functions are called “CRUD” functions.

CRUD is an acronym that stands for CREATE, READ, UPDATE, and DELETE. For example, in a SQL database, the CRUD functions are used to interact with or manipulate the data stored in the database.

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

API Tools and Resources

Whether you’re trying to learn how APIs interact with applications or controllers, need to test code and outcomes, or want to become a full-time developer, 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.

Introduction to Postman

Interaction with a software controller using RESTful APIs.

It also discussed being able to test code to see if the desired outcomes are accomplished when executing the code.

Keep in mind that APIs are software interfaces into an application or a controller.

Many APIs require authentication.

This means that such an API is considered just like any other device to **which a user needs to authenticate to gain access to utilize the APIs.**

A developer who is authenticated has access to making changes using the API, which can impact that application.

This means if a **REST API call is used to delete data**, that **data will be removed** from the application or controller just as if a user logged into the device via the CLI and deleted it.

It is best practice to use a test lab or the Cisco DevNet sandbox while learning or practicing any of these concepts to avoid accidental impact to a production or lab environment.

The Postman application has various sections that you can interact with. The focus here is on using the Builder portion of the dashboard. The following sections are the ones that require the most focus and attention:

- History
- Collections
- New Tab
- URL bar

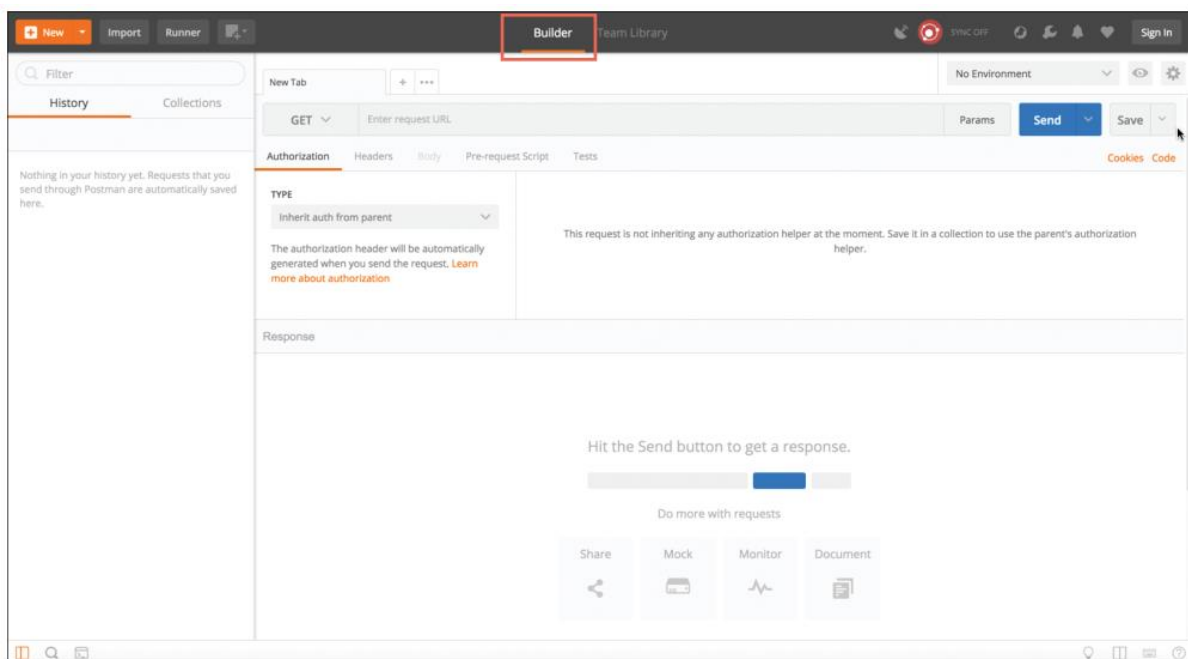


Figure 28-2 *Postman Dashboard*

The History tab shows a list of all the recent API calls made using Postman. Users have the option to clear their entire history at any time if they want to remove the complete list of API calls that have been made. This is done by clicking the Clear All link at the top of the Collection window (see Figure

28-3). Users also have the ability to remove individual API calls from the history list by simply hovering the mouse over an API call and clicking the trash can icon in the submenu that pops up.

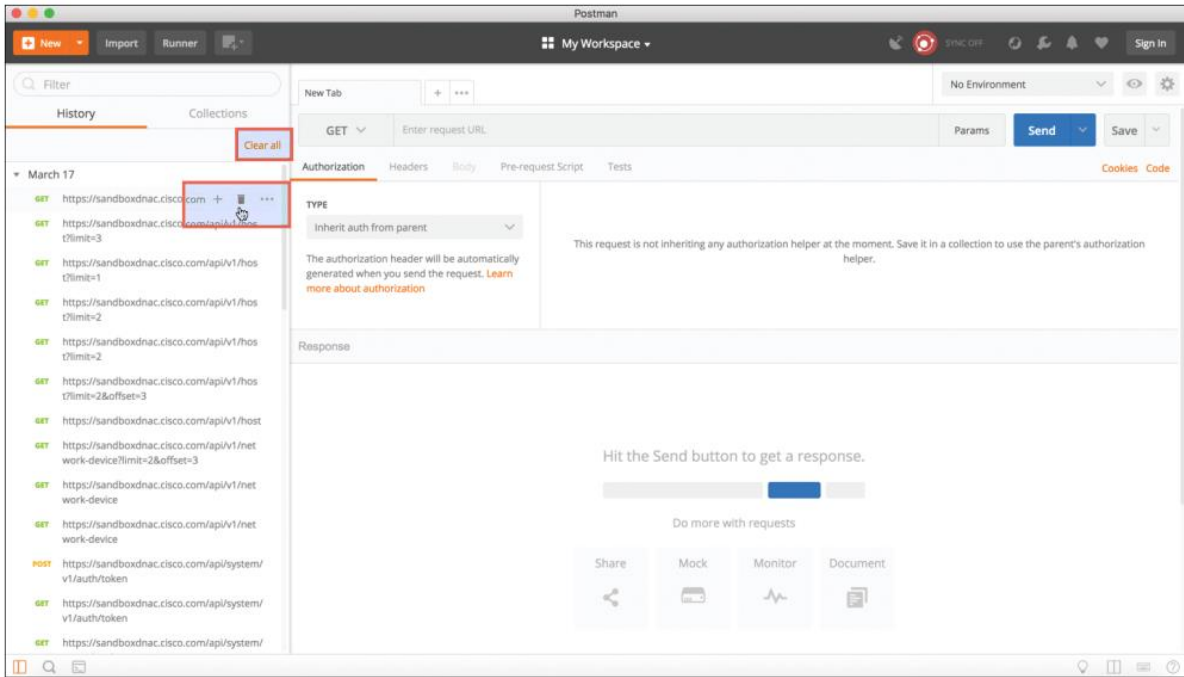


Figure 28-3 Clearing the Postman API History

API calls can be stored in groups, called *collections*, that are specific to a structure that fits the user’s needs. Collections can follow any naming convention and appear as a folder hierarchy. For example, it’s possible to have a collection called DNA-C to store all the Cisco DNA Center API calls. Saving API calls to a collection helps during testing phases as the API calls can easily be found and sorted.

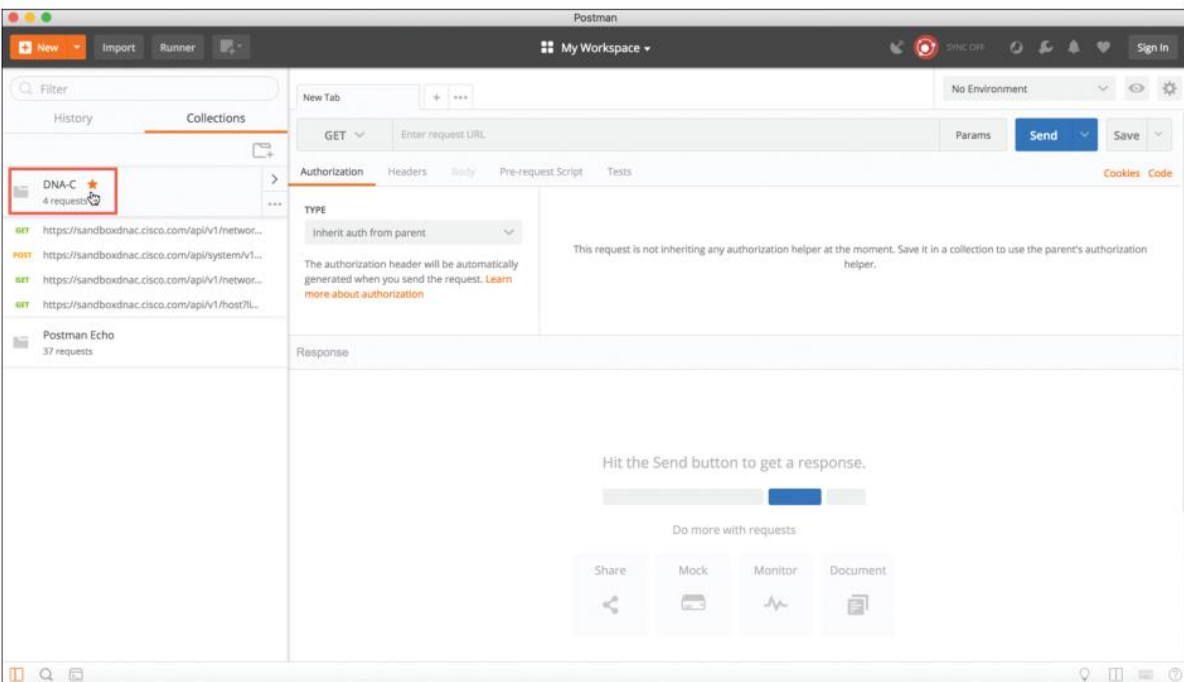


Figure 28-4 A Favorite Postman Collection

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. For example, a user can have one tab open with API calls interacting with the Cisco DNA Center controller and another tab open that is interacting with a completely different platform, such as a Cisco Nexus switch. 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

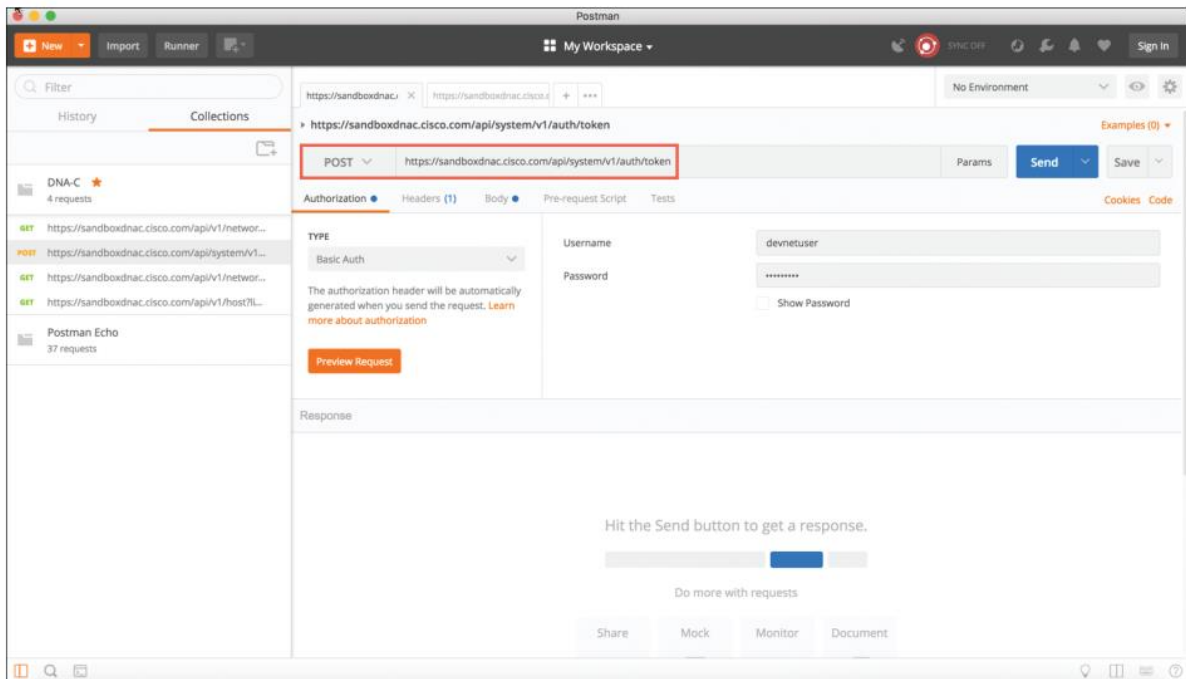


Figure 28-5 *Postman URL Bar with Cisco DNA Center Token API Call*

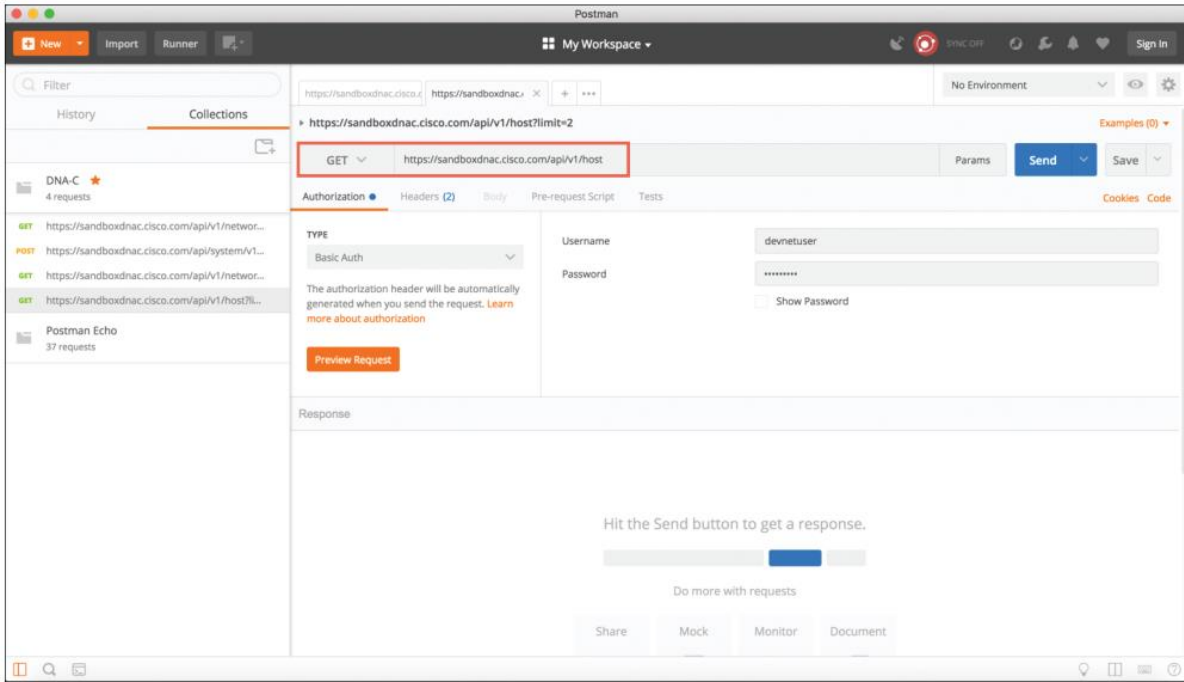


Figure 28-6 Postman URL Bar with Cisco DNA Center Host API Call

Cisco DNA Center APIs

The Cisco DNA Center controller expects all incoming data from the REST API to be in JSON format.

It is also important to note that 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.

Just as users do when logging in to a device via the CLI, if secured properly, they should be prompted for login credentials.

The same method applies to using an API to authenticate to software. 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.

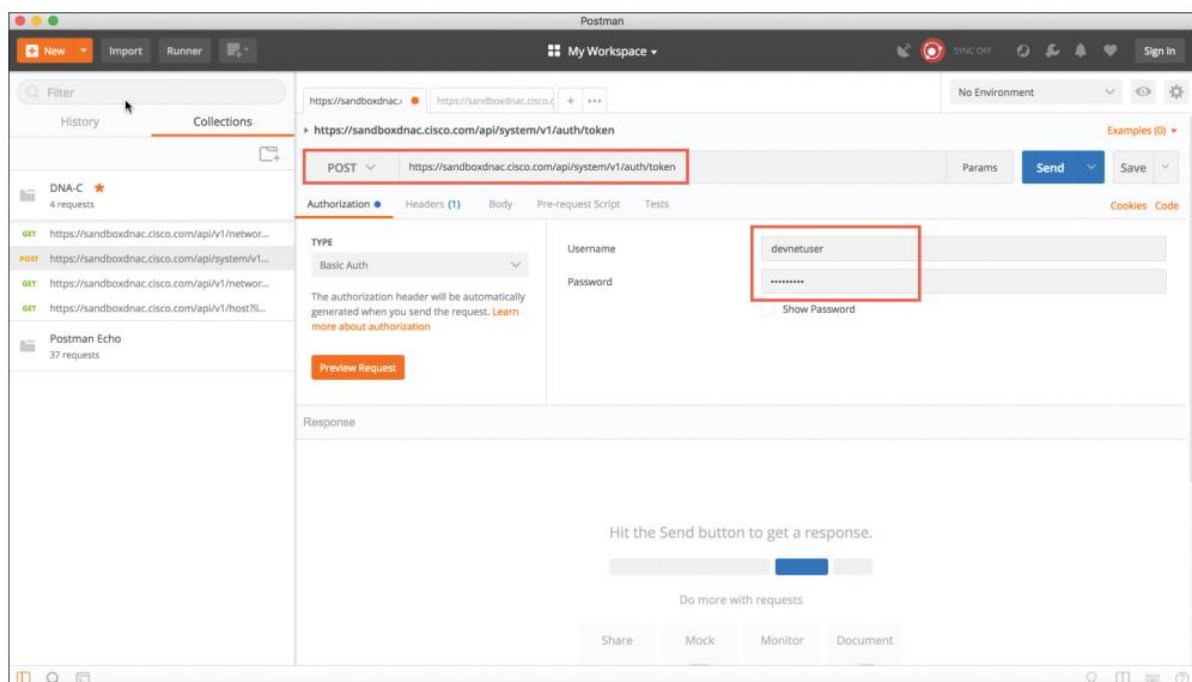


Figure 28-7 Setting Up Postman to Authenticate with the Cisco DNA Center Controller

You need a token for any future API calls to the Cisco DNA Center controller. When you are successfully authenticated to the Cisco DNA Center controller, you receive a token that contains a string that looks similar to the following:

```
"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWwiOiI1YTU4Y2QzN2UwNWJiYTAwOGVmNjJiOTIiLCJhdXRoU291cmNlIjoiaW50ZXJyZWwiLCJ0ZW5hbnR0YXV1IjoiVE5UMCIsInJvbGVzIjpbIjVhMzE1MTYwOTA5MGZiYTY5OGlyZjViNyJdLCJ0ZW5hbnRjZCI6IjVhMzE1MTkZTA1YmJhMDA4ZWY2MmYyYSIsImV4cCI6MTUyMTQ5NzI2NCwidXNlcm5hbWUwOiIjZXZuZXR1c2VyIn0.tgAJfLc10aUwaJCX6lfzjPG7Om2x97oiTlozUpAzomM"
```

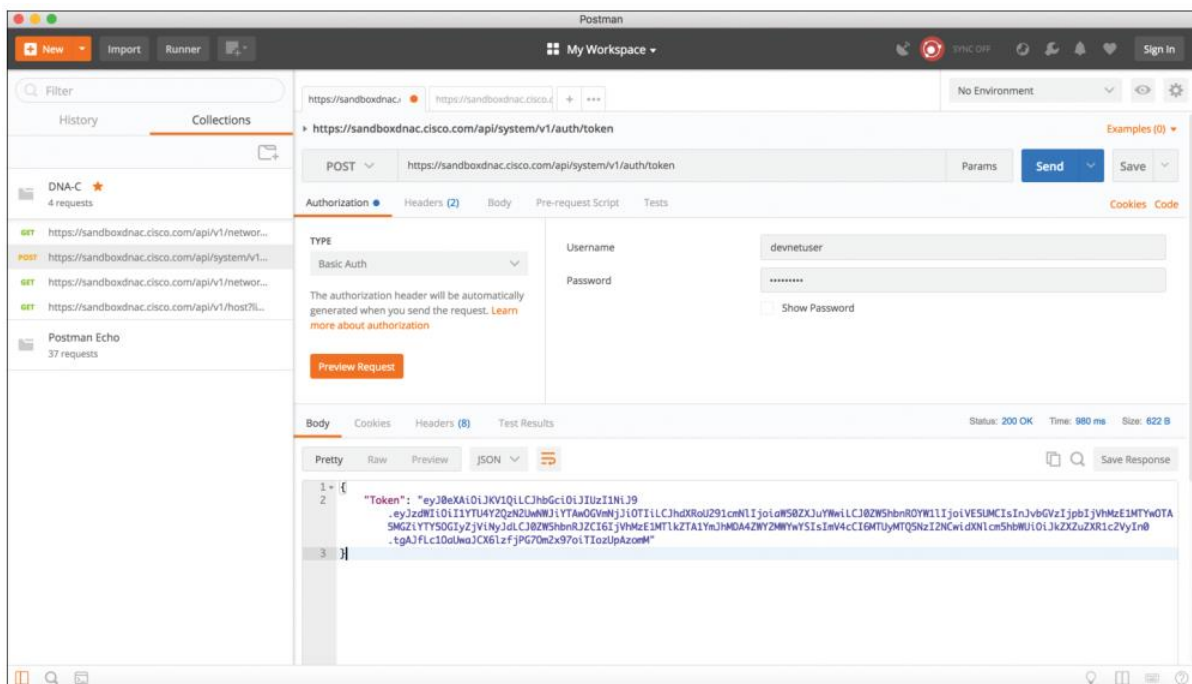


Figure 28-8 Cisco DNA Center POST Operation

You can see in the top right of the screen shown in Figure 28-8 that the received HTTP status code from the Cisco DNA Center controller is 200 OK.

You can tell that the HTTP status code 200 means that the API call completed successfully. In addition, you can see how long it took to process the HTTP POST request: 980 ms.

Now we can take a look at some of the other available API calls. The first API call that is covered in this section is the Network Device API, which allows users to retrieve a list of devices that are currently in inventory that are being managed by the Cisco DNA Center controller.

- Step 1. Copy the token you received earlier and click a new tab in Postman.
- Step 2. In the URL bar enter https://sandboxnac.cisco.com/api/v1/network-device to target the Network Device API.
- 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 using the Network Device API

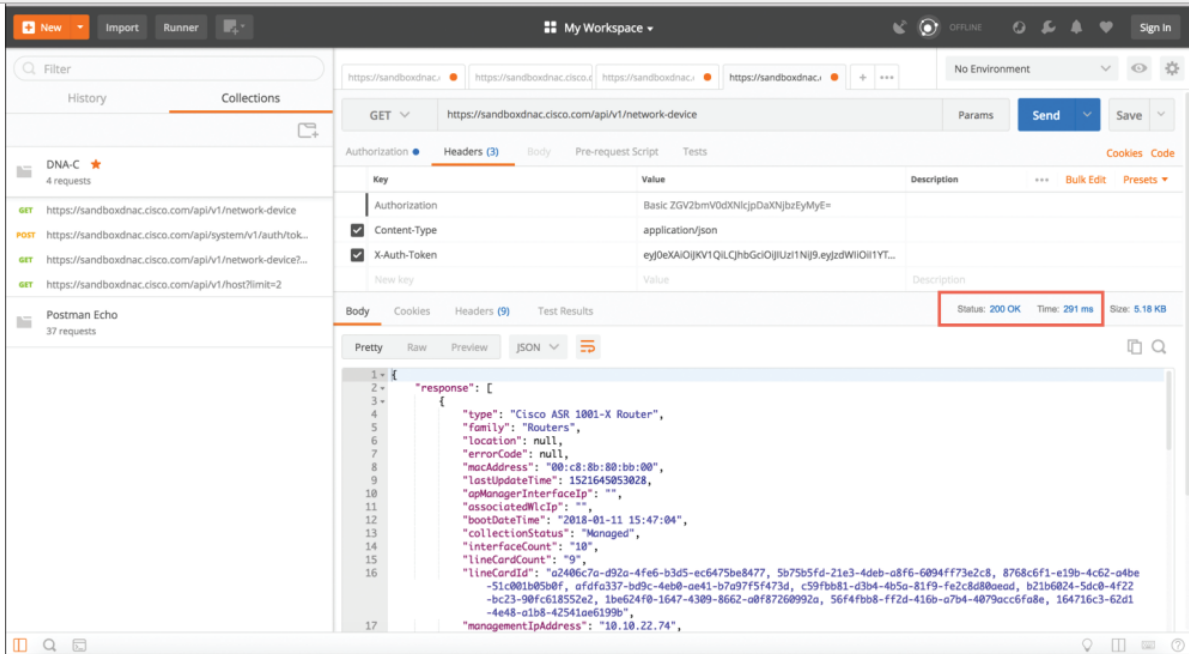
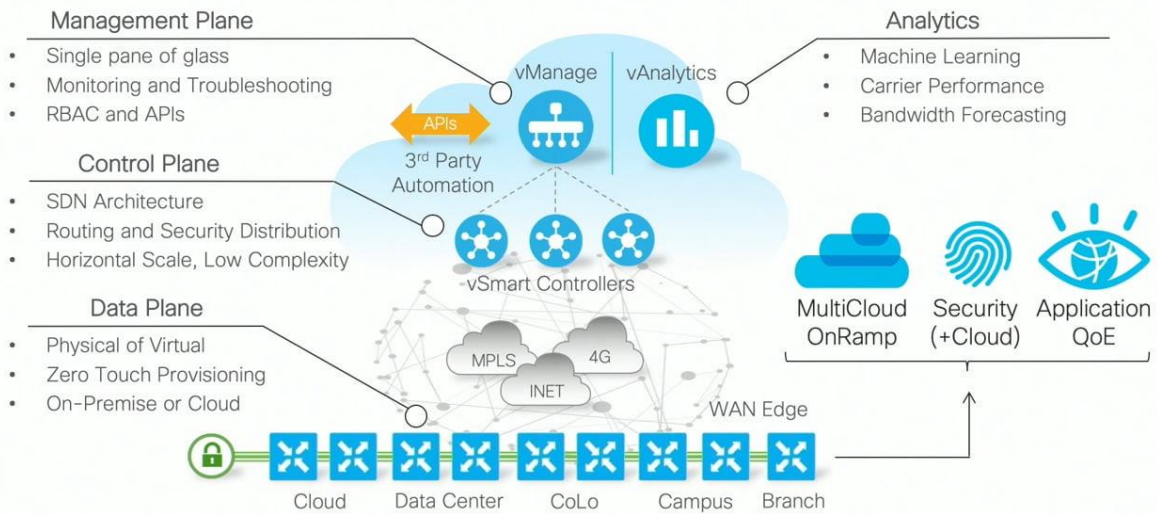


Figure 28-9 Postman Setup for Retrieving the Network Device Inventory with an API Call

Based on the response received from the Cisco DNA Center controller, you can see the HTTP status code 200 OK, and you can also see that a device inventory was received, in JSON format.

Cisco vManage APIs

Cisco SD-WAN: Cloud SDN Architecture



Cisco SD-WAN APIs is a bit different from using the Cisco DNA Center APIs, but the two processes are quite similar.

As when using a Cisco DNA Center API, 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!.

Postman environment set up for the Cisco SD-WAN API calls—specifically, the Authentication API

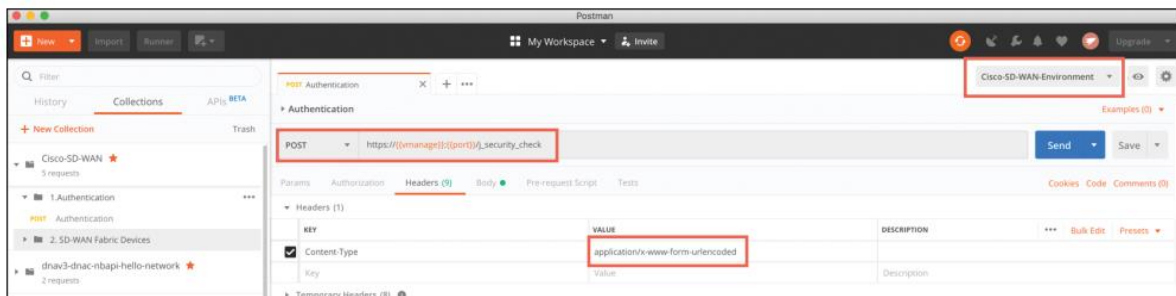


Figure 28-11 Cisco vManage Authentication API Setup for Postman

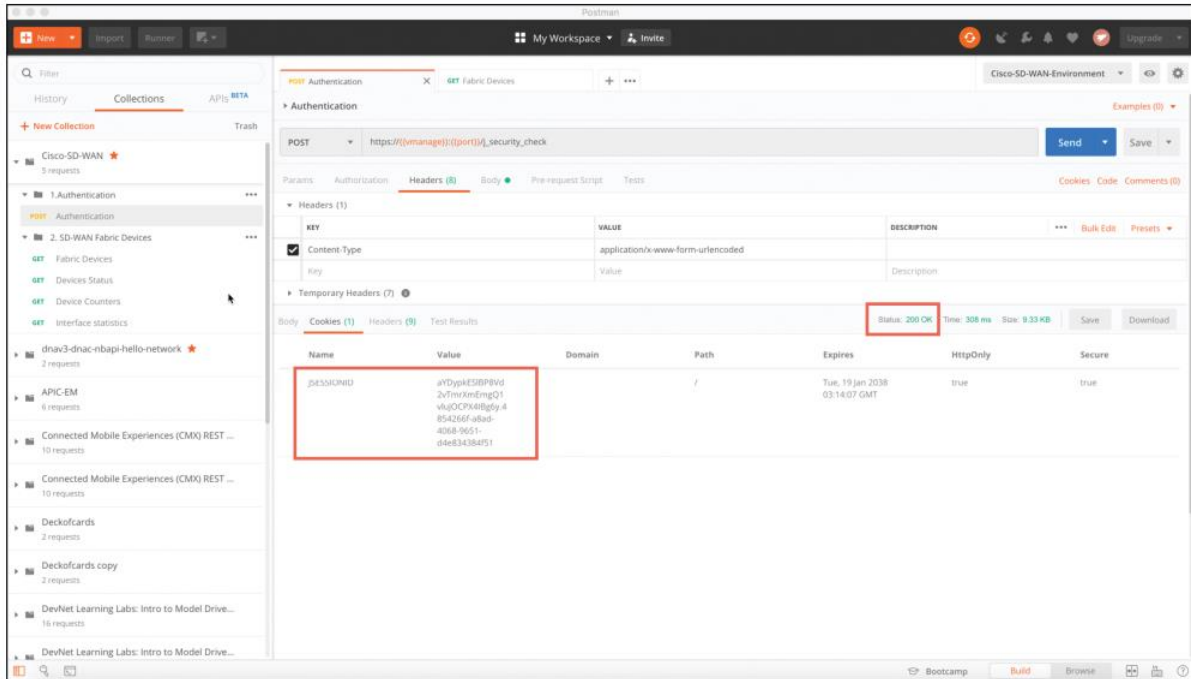


Figure 28-12 Successful HTTP POST to Cisco vManage Authentication API

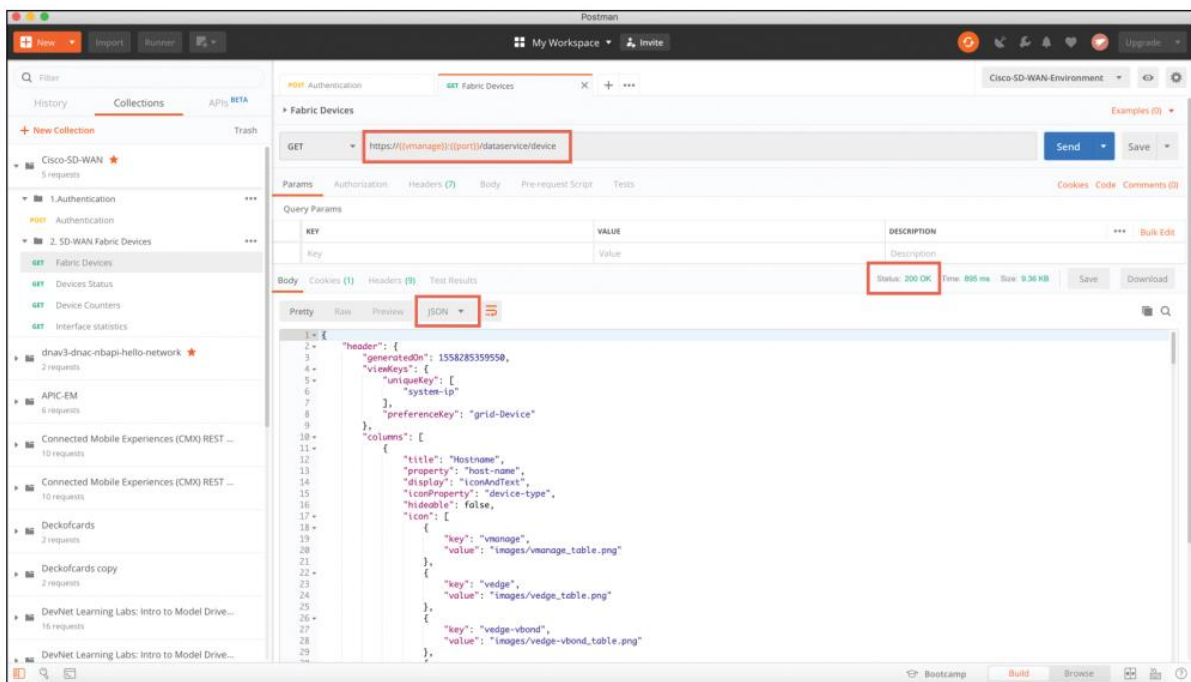


Figure 28-13 Successful HTTP GET to the Cisco vManage Fabric Device API

If you scroll down in the response, you can see a list of devices under the “data” key received from the API call. This list contains a series of information about each fabric device within Cisco vManage. Some of the information you can see in Figure 28-14 is as follows:

- Device ID
- System IP
- Host name

- Reachability
- Status
- Device type
- Site ID

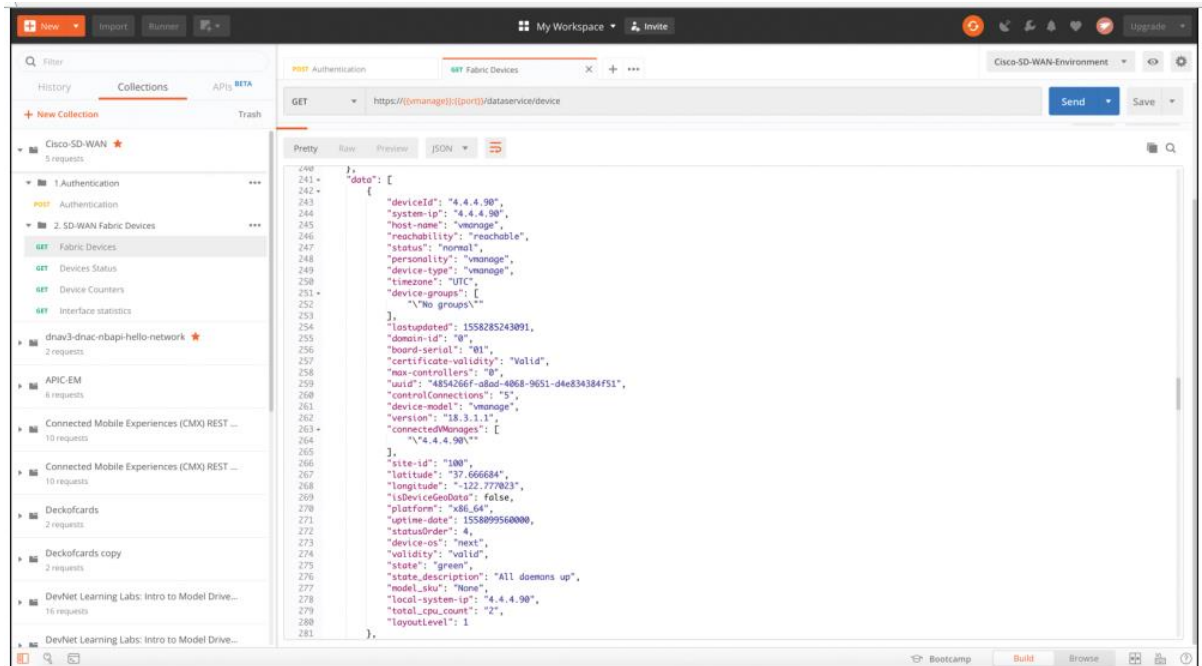


Figure 28-14 Data Received with a Successful HTTP GET to the Cisco vManage Fabric Device API

As you can see, a single API call has the power to gather a significant amount of information. How the data is used is up to the person making the API calls and collecting the data.

Detailed steps for setting up the Postman environment for Cisco SD-WAN are available at <https://developer.cisco.com/sdwan/>


Data Formats (XML, JSON)

Now that the Postman dashboard has been shown, it's time to discuss two of the most common data formats that are used with APIs.

Text-file formats that can be used to **store structured data that can be handy for embedded and Web applications.**

1. XML:

XML (standard)

Status	Published
Year started	1996; 24 years ago
Latest version	1.1 (Second Edition) September 29, 2006; 13 years ago
Editors	Tim Bray Jean Paoli C. M. Sperberg-McQueen Eve Maler François Yergeau John Cowan
Related standards	XML Schema
Domain	Data serialization
Abbreviation	XML
Website	w3.org/TR/xml11 

Extensible Markup Language (XML) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable.

```
<?xml version="1.0" encoding="UTF-8"?>
```

Example:

```
&lt;book id="bk101"&gt;
&lt;author&gt;Gambardella, Matthew&lt;/author&gt;
&lt;title&gt;XML Developer's Guide&lt;/title&gt;
&lt;genre&gt;Computer&lt;/genre&gt;
&lt;price&gt;44.95&lt;/price&gt;
&lt;publish_date&gt;2000-10-01&lt;/publish_date&gt;
&lt;description&gt;An in-depth look at creating applications
with XML.&lt;/description&gt;
&lt;/book&gt;
```

2. JSON

JavaScript Object Notation (JSON) is used with JavaScript, of course. It will be familiar to Web developers that use it for client/server communication.

JSON uses name/value pairs

JSON was **first standardized in 2013**

```
{
  "books": [
    {
      "id": "bk102",
      "author": "Crockford, Douglas",
      "title": "JavaScript: The Good Parts",
      "genre": "Computer",
      "price": 29.99,
      "publish_date": "2008-05-01",
      "description": "Unearthing the Excellence in JavaScript"
    }
  ]
}
```

3. YAML

YAML stands for YAML Ain't Markup Language. It uses line and whitespace delimiters instead of explicitly marked blocks that could span one or more lines like XML and JSON. This approach is used in many programming languages, such as Python.

It is commonly used for **configuration files** and in **applications** where **data is being stored or transmitted**.

Example1: Ansible, Flask

YAML

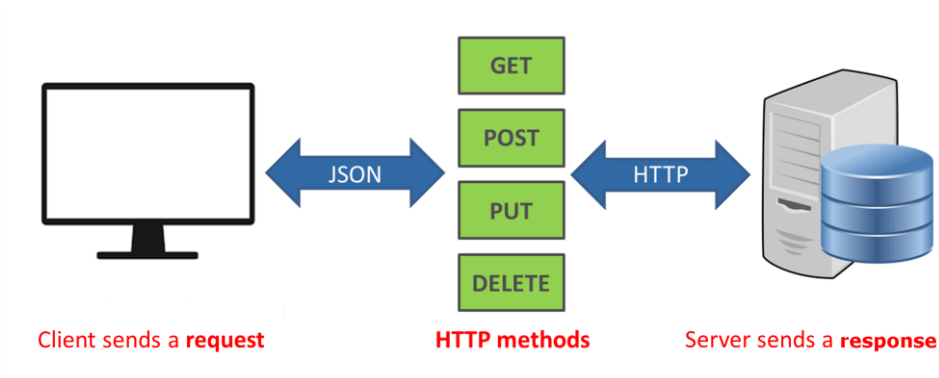
Filename extensions	.yaml , .yml
Internet media type	Not registered
Initial release	11 May 2001; 19 years ago
Latest release	1.2 (Third Edition) (1 October 2009; 10 years ago)
Type of format	Data interchange
Open format?	Yes
Website	yaml.org

Example2:

```
books:
  - id: bk102
    author: Crockford, Douglas
    title: 'JavaScript: The Good Parts'
    genre: Computer
    price: 29.99
    publish_date: !!str 2008-05-01
    description: Unearthing the Excellence in JavaScript
```

Now that the XML and JSON data formats have been explained, it is important to circle back to actually 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. However, many users don’t know what this error actually means.



HTTP STATUS CODES

2xx Success

200 Success / OK

3xx Redirection

301 Permanent Redirect

302 Temporary Redirect

304 Not Modified

4xx Client Error

401 Unauthorized Error

403 Forbidden

404 Not Found

405 Method Not Allowed

5xx Server Error

501 Not Implemented

502 Bad Gateway

503 Service Unavailable

504 Gateway Timeout

Data Models and Supporting Protocols

1. Yet Another Next Generation (YANG) modeling language
2. Network Configuration Protocol (NETCONF)
3. RESTCONF

Data models describe the things you can **configure, monitor, and the actions you can perform on a network device.**

1. Yet Another Next Generation (YANG) modeling language

YANG(Yet Another Next Generation) is a data modeling language for the definition of data sent over network management protocols such as the NETCONF and RESTCONF.

Many network management protocols have associated data modeling languages. **SNMP** is widely used for fault handling and monitoring. However, it is not often used for configuration changes. CLI scripting is used more often than other methods. The data modeling language associated with **SNMP** was called the **Structure of Management Information (SMI)**. In the late 1990s, a project was started to create a replacement for SMIv2, which was called SMIng, which was failed.

YANG data models are an alternative to SNMP MIBs and are becoming the standard for data definition languages.

YANG, which is defined in RFC 6020, uses data models. Data models are used to describe whatever can be **configured on a device, everything that can be monitored on a device, and all the administrative actions that can be executed on a device, such as resetting counters or rebooting the device.** This includes all the notifications that the device is capable of generating. All these variables can be represented within a YANG model. Data models are very powerful in that they create a uniform way to describe data, which can be beneficial across vendors' platforms.

Data models allow network operators to configure, monitor, and interact with network devices holistically across the entire enterprise environment.

YANG models use a tree structure. Within that structure, the models are similar in format to XML and are constructed in modules. These modules are hierarchical in nature and contain all the different data and types that make up a YANG device model. YANG models make a clear distinction between configuration data and state information. 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. However, the operational interface state cannot be changed; for example, if the options are only up or down, it is either up or down, and nothing else is possible.

YANG 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;
}
}
}
}
}
}
}
}
}

```

2. NETCONF

NETCONF, defined in RFC 4741 and RFC 6241, is an IETF (Internet Engineering Task Force) 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), Simple Object Access Protocol (SOAP).

One of the most important differences is that SNMP can't distinguish between configuration data and operational data, but NETCONF can.

The following is a list of some of the common use cases for NETCONF:

- 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

Table 28-6 Differences Between SNMP and NETCONF

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 element from RFC 4741. This NETCONF output can be read as follows: There is an XML list of users named users. In that list, there are individual users named Dave, Rafael, and Dirk.

```

<rpc-reply message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">

```

```

<data>
<top xmlns="http://example.com/schema/1.2/config">
<users>
<user>
<name>Dave</name>
</user>
<user>
<name>Rafael</name>
</user>
<user>
<name>Dirk</name>
</user>
</users>
</top>
</data>
</rpc-reply>

```

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.

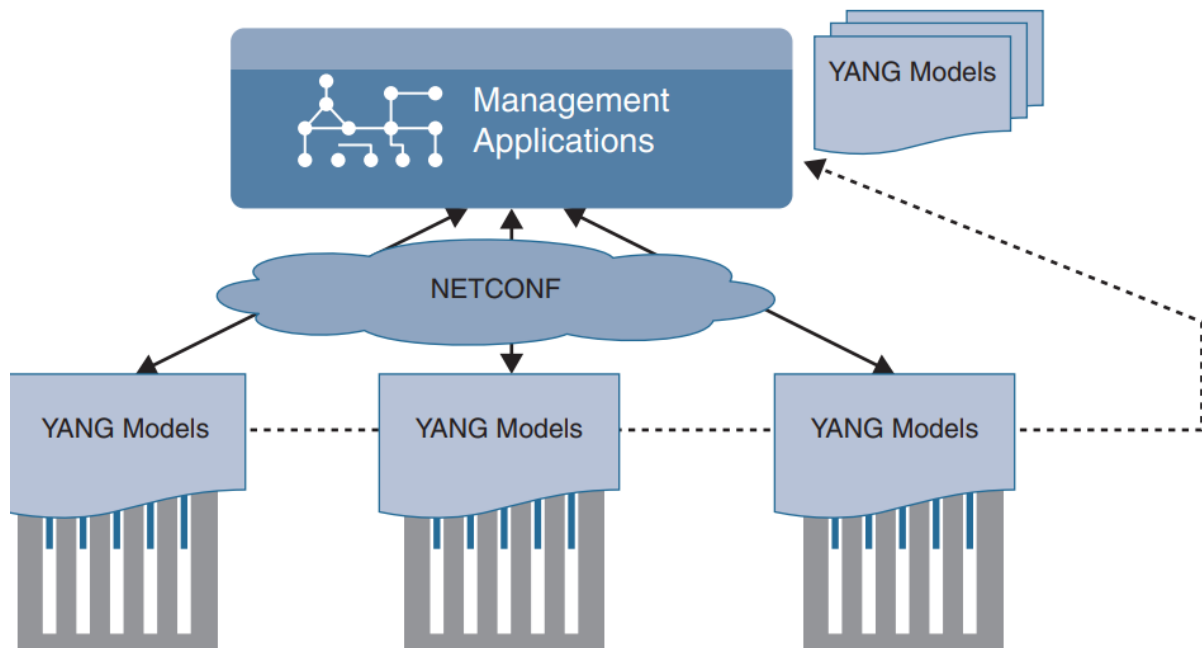


Figure 28-15 NETCONF/YANG Interfacing with Management Applications

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.

Table 28-7 NETCONF Operations

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

Now that we've looked at the basics of NETCONF and XML, let's examine some actual examples of a NETCONF RPC message. Below example of an OSPF NETCONF RPC message that provides the OSPF routing configuration of an IOS XE device.

```
<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/ncs/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>
```

3. RESTCONF

RESTCONF, defined in RFC 8040, is used to programmatically interface with data defined in YANG models while also using the datastore concepts defined in NETCONF.

There is a common misconception that RESTCONF is meant to replace NETCONF, but this is not the case. Both are very common methods used for programmability and data manipulation. In fact, 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.

```
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"  
}
```

QUIZ#

1. True or false: Python is considered one of the most difficult programming languages to learn and adopt.
 - a. True
 - b. False

2. To authenticate with Cisco's DNA Center, which type of HTTP request method must be used?
 - a. PUT
 - b. PATCH
 - c. GET
 - d. POST
 - e. HEAD

3. What does CRUD stand for?
 - a. CREATE, RESTORE, UPDATE, DELETE
 - b. CREATE, READ, UPDATE, DELETE
 - c. CREATE, RETRIEVE, UPDATE, DELETE
 - d. CREATE, RECEIVE, UPLOAD, DOWNLOAD
 - e. CREATE, RECEIVE, UPLOAD, DELETE

4. When using the Cisco vManage Authentication API, what is the Headers ContentType that is used?
 - a. MD5
 - b. X-Auth-Token
 - c. SSH
 - d. x-www-form-urlencoded
 - e. JSON

5. Which of the following is in JSON data format?
 - a.
{


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

b.

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

c.

```
root  
Jason  
Jamie  
Luke
```

d.

```
[users[root|Jason|Jamie|Luke]]
```

6. What is the HTTP status code for Unauthorized?

- a. 201
- b. 400

c. 401

d. 403

e. 404

7. In Python, why would you use three quotation marks in a row? (Choose two.)

a. To begin a multiple-line string

b. To start a function

c. To represent a logical OR

d. To end a multiple-line string

e. To call a reusable line of code

8. Which of the following is a Python dictionary?

a.

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

b.

```
[users[root|Jason|Jamie|Luke]]
```

c.

```
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"]
```

d.

```
print(dnac_devices)
```

9. Which of the following are Python functions? (Choose two.)

a.

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

b.

```
[users[root|Jason|Jamie|Luke]]
```

c.

```
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"]
```

d.

```
print(dnac_devices)
```

10. When using the Cisco DNA Center Token API, what authentication method is used?

a. MD5

b. X-Auth-Token

c. SSH

d. Basic authentication

e. JSON

11. What is the DevNet Community page used for? (Choose two.)

a. To ask questions

- b. To exchange code
- c. To access learning labs
- d. To access DevNet ambassadors and evangelists
- e. To get news on local DevNet events

12. When using GitHub, what is the purpose of a repository? (Choose three.)

- a. Provides a place to store a developer's code
- b. Provides a place to store music and photos
- c. Gives the option to share a developer's code with other users
- d. Provides documentation on code examples
- e. Offers a sandbox to test custom code

13. Why is using the command-line interface (CLI) to configure a large number of devices considered difficult to scale? (Choose two.)

- a. The CLI is prone to human error and misconfiguration.
- b. The CLI is quick and efficient for configuring many devices simultaneously.
- c. Telnet access to the CLI is best practice.
- d. The command line is used on a device-by-device basis.
- e. Using APIs is considered a legacy method of configuration.

14. Which of the following are part of the YANG model? (Choose two.)

- a. Type
- b. Leaf
- c. Container
- d. String
- e. Method

ANSWERS:

- 1 B
- 2 D
- 3 B

4 D

5 A

6 C

7 A, D

8 A

9 C, D

10 D

11 A, D

12 A, C, D

13 A, D

14 B, C

CISCO IOS EMBEDDED EVENT MANAGER (EEM)

Embedded Event Manager (EEM) is a technology on Cisco Routers that lets you run scripts or commands **when a certain event happens**.

1a. SYSLOG EVENTS

Syslog messages are the messages that you see by default on your console. Interfaces going up or down, OSPF neighbors that disappear and such are all syslog messages.

EEM can take action when one of these messages show up.

```
ROUTER(config)#  
  
event manager applet INTERFACE_DOWN  
  
event syslog pattern "Interface FastEthernet0/0, changed state to down"  
  
action 1.0 cli command "enable"  
  
action 2.0 cli command "conf term"  
  
action 3.0 cli command "interface fa0/0"  
  
action 4.0 cli command "no shut"
```

The applet is called "INTERFACE_DOWN" and the event is a syslog pattern that matches the text when an interface goes down. When this occurs, we run a number of commands. What happens is that whenever someone shuts the interface, EEM will do a "no shut" on it.

To demonstrate that this works I'll enable a debug:

```
ROUTER#debug event manager action cli  
  
Debug EEM action cli debugging is on
```

This will show the commands that EEM runs when the event occurs. Let's do a shut on that interface:

```
ROUTER (config)#interface FastEthernet 0/0
```

```
ROUTER (config-if)#shutdown
```

Within a few seconds you will see this:

```
ROUTER #  
  
%LINK-5-CHANGED: Interface FastEthernet0/0, changed state to administratively down  
  
%LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet0/0, changed state to down  
  
%HA_EM-6-LOG: INTERFACE_DOWN : DEBUG(cli_lib) : : CTL : cli_open called.  
  
%HA_EM-6-LOG: INTERFACE_DOWN : DEBUG(cli_lib) : : OUT : R2>  
  
%HA_EM-6-LOG: INTERFACE_DOWN : DEBUG(cli_lib) : : IN : R2>enable  
  
%HA_EM-6-LOG: INTERFACE_DOWN : DEBUG(cli_lib) : : OUT : R2#  
  
%HA_EM-6-LOG: INTERFACE_DOWN : DEBUG(cli_lib) : : IN : R2#conf term  
  
%HA_EM-6-LOG: INTERFACE_DOWN : DEBUG(cli_lib) : : OUT : Enter configuration commands, one  
per line. End with CNTL/Z.  
  
%HA_EM-6-LOG: INTERFACE_DOWN : DEBUG(cli_lib) : : OUT : R2(config)#  
  
%HA_EM-6-LOG: INTERFACE_DOWN : DEBUG(cli_lib) : : IN : R2(config)#interface fa0/0  
  
%HA_EM-6-LOG: INTERFACE_DOWN : DEBUG(cli_lib) : : OUT : R2(config-if)#  
  
%HA_EM-6-LOG: INTERFACE_DOWN : DEBUG(cli_lib) : : IN : R2(config-if)#no shut  
  
%HA_EM-6-LOG: INTERFACE_DOWN : DEBUG(cli_lib) : : OUT : R2(config-if)#  
  
%HA_EM-6-LOG: INTERFACE_DOWN : DEBUG(cli_lib) : : CTL : cli_close called.  
  
%LINK-3-UPDOWN: Interface FastEthernet0/0, changed state to up
```

```
%LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet0/0, changed state to up
```

The interface went down, EEM runs the commands and the interface is up again. Simple but I think this is a good example to demonstrate how EEM works.

1b. OSPF ADJACENCY CHANGES

The next example is perhaps useful. Whenever the OSPF adjacency disappears you will see a syslog message on your console. We'll use this message as the event and once it occurs, we enable OSPF adjacency debugging and send an e-mail:

```
ROUTER (config)#
event manager applet OSPF_DOWN

event syslog pattern "Nbr 192.168.12.1 on FastEthernet0/0 from FULL to DOWN"

action 1.0 cli command "enable"

action 2.0 cli command "debug ip ospf adj"

action 3.0 mail server "smtp.networkjourney.com" to "sagardhawan@networkjourney.com" from
"R2@networkjourney.com" subject "OSPF IS DOWN" body "Please fix OSPF"
```

The event that I used is a syslog message that should look familiar. The first two actions are executed on the CLI but the third action is for the e-mail. It will send a message to sagardhawan@networkjourney.com through SMTP-server "smtp.networkjourney.com".

Let's give it a try. I have to enable another debug if I want to see the mail action:

```
ROUTER #debug event manager action mail

Debug EEM action mail debugging is on
```

Once the OSPF neighbor adjacency is established, I'll shut the interface on one of the routers so it breaks:


```
R1(config)#interface FastEthernet 0/0
```

```
R1(config-if)#shutdown
```

And this is what you'll see:

```
ROUTER #
```

```
Translating "smtp.networkjourney.com"...domain server (255.255.255.255)
```

```
%OSPF-5-ADJCHG: Process 1, Nbr 192.168.12.1 on FastEthernet0/0 from FULL to DOWN, Neighbor Down: Dead timer expired
```

```
%HA_EM-6-LOG: OSPF_DOWN : DEBUG(cli_lib) : : CTL : cli_open called.
```

```
%HA_EM-6-LOG: OSPF_DOWN : DEBUG(cli_lib) : : OUT : R2>
```

```
%HA_EM-6-LOG: OSPF_DOWN : DEBUG(cli_lib) : : IN : R2>enable
```

```
%HA_EM-6-LOG: OSPF_DOWN : DEBUG(cli_lib) : : OUT : R2#
```

```
%HA_EM-6-LOG: OSPF_DOWN : DEBUG(cli_lib) : : IN : R2#debug ip ospf adj
```

```
%HA_EM-6-LOG: OSPF_DOWN : DEBUG(cli_lib) : : OUT : OSPF adjacency events debugging is on
```

```
%HA_EM-6-LOG: OSPF_DOWN : DEBUG(cli_lib) : : OUT : R2#
```

```
%HA_EM-6-LOG: OSPF_DOWN : DEBUG(smtp_lib) : smtp_connect_attempt: 1
```

```
OSPF: Build router LSA for area 0, router ID 192.168.12.2, seq 0x8000000B, process 1
```

```
OSPF: No full nbrs to build Net Lsa for interface FastEthernet0/0
```

```
OSPF: Build network LSA for FastEthernet0/0, router ID 192.168.12.2
```

```
OSPF: Build network LSA for FastEthernet0/0, router ID 192.168.12.2
```

```
%HA_EM-6-LOG: OSPF_DOWN : DEBUG(smtp_lib) : fh_smtp_connect failed at attempt 1
```

```

Translating "smtp.networkjourney.com"...domain server (255.255.255.255)

%HA_EM-6-LOG: OSPF_DOWN : DEBUG(smtp_lib) : smtp_connect_attempt: 2

%HA_EM-6-LOG: OSPF_DOWN : DEBUG(smtp_lib) : fh_smtp_connect callback timer is awake

%HA_EM-3-FMPD_SMTP: Error occurred when sending mail to SMTP server:
smtp.networkjourney.com: timeout error

%HA_EM-6-LOG: OSPF_DOWN : DEBUG(cli_lib) : : CTL : cli_close called.

```

My router isn't connected to the Internet but you can see it's trying to contact the SMTP server and send an e-mail. It also enabled the OSPF adjacency debug thanks to the CLI commands.

2. CLI Events

The previous two examples used syslog messages as the event but you can also take action based on commands that are used on the CLI.

```

ROUTER (config)#

event manager applet SHOW_RUN_NO_INTERFACES

event cli pattern "show run" sync yes

action 1.0 cli command "enable"

action 2.0 cli command "show run | exclude interface"

action 3.0 puts "$_cli_result"

action 4.0 set $_exit_status "0"

```

As you can see above the event is a CLI pattern. the "sync yes" parameter is required, this tells EEM to run the script before running the "show run" command. When the script is done, it sets the exit status to 0. Basically this means that whenever someone uses the "show run" command, the script will run "show run | exclude interface" instead and gives you the output.

Let's see what the result is...

```
ROUTER #show running-config
```

```
Building configuration...
```

You will see the output of the running configuration and if you left the debug on, you'll see what EEM is doing behind the scenes:

```
ROUTER #
```

```
%HA_EM-6-LOG: SHOW_RUN_NO_INTERFACES : DEBUG(cli_lib) : : CTL : cli_open called.
```

```
%HA_EM-6-LOG: SHOW_RUN_NO_INTERFACES : DEBUG(cli_lib) : : OUT : R2>
```

```
%HA_EM-6-LOG: SHOW_RUN_NO_INTERFACES : DEBUG(cli_lib) : : IN : R2>enable
```

```
%HA_EM-6-LOG: SHOW_RUN_NO_INTERFACES : DEBUG(cli_lib) : : OUT : R2#
```

```
%HA_EM-6-LOG: SHOW_RUN_NO_INTERFACES : DEBUG(cli_lib) : : IN : R2#show run | exclude  
interface
```

```
%HA_EM-6-LOG: SHOW_RUN_NO_INTERFACES : DEBUG(cli_lib) : : OUT : Building configuration...
```

Somewhere further down the running-config you can see that the lines with "interface" in them were removed:

```
!
```

```
ip address 192.168.12.2 255.255.255.0
```

```
duplex auto
```

```
speed auto
```

3. Interface Events

You have seen syslog and CLI pattern events, but we have some others. What about interface counters? It might be useful to perform an action when some interface counters have a certain value. Here's an example:

```
ROUTER #show interfaces fastEthernet 0/0 | incl load

reliability 255/255, txload 1/255, rxload 1/255
```

Let's create a script that does something when the interface load hits a certain value. To make this work, it's best to change the load interval of the interface first:

```
ROUTER (config)#interface FastEthernet 0/0

ROUTER (config-if)#load-interval 30
```

By using this command, the router will calculate the load of the interface every 30 seconds, the default is 5 minutes. Let's create the script:

```
ROUTER (config)#

event manager applet INTERFACE_LOAD

event interface name FastEthernet0/0 parameter rxload entry-op gt entry-val 10 entry-type value
poll-interval 10

action 1.0 syslog priority informational msg "INTERFACE OVERLOADED"
```

This event is a bit harder to read...when the rx load of the interface is above 10/255 then we will take action. Every 10 seconds we will check if we reached this value or not. When the event occurs, a syslog message is produced.

To demonstrate this we'll send some packets from R1 towards R2:

```
ROUTER #ping 192.168.12.2 repeat 9999999 size 15000 timeout 0
```

Once the interface rx load is above 10 you'll see the following message on the console:

```
ROUTER #  
  
%HA_EM-6-LOG: INTERFACE_LOAD: INTERFACE OVERLOADED
```

4. Scheduling Events

Instead of launching actions based on syslog or CLI messages we can also use scheduled tasks. This means that you can run actions every X minutes / hours / days etc. Here's an example:

```
ROUTER (config)#  
  
event manager applet TIMER  
  
event timer watchdog time 60  
  
action 1.0 cli command "enable"  
  
action 2.0 cli command "write memory"  
  
action 3.0 syslog priority informational msg "Configuration has been saved"
```

This script runs every 60 seconds and runs the "write memory" command. Once it's done, it will produce a syslog message. After waiting for 60 seconds we'll see this:

```
ROUTER #  
  
%HA_EM-6-LOG: TIMER : DEBUG(cli_lib) : : CTL : cli_open called.  
  
%HA_EM-6-LOG: TIMER : DEBUG(cli_lib) : : OUT : R2>  
  
%HA_EM-6-LOG: TIMER : DEBUG(cli_lib) : : IN : R2>enable  
  
%HA_EM-6-LOG: TIMER : DEBUG(cli_lib) : : OUT : R2#  
  
%HA_EM-6-LOG: TIMER : DEBUG(cli_lib) : : IN : R2#write memory  
  
%HA_EM-6-LOG: TIMER : DEBUG(cli_lib) : : OUT : Building configuration...
```

```
%HA_EM-6-LOG: TIMER : DEBUG(cli_lib) : : OUT : [OK]

%HA_EM-6-LOG: TIMER : DEBUG(cli_lib) : : OUT : R2#

%HA_EM-6-LOG: TIMER: Configuration has been saved

%HA_EM-6-LOG: TIMER : DEBUG(cli_lib) : : CTL : cli_close called.
```

5. Other Events and Actions

You have seen a couple of events and actions but EEM has a lot of options. Here's a list to give you some ideas:

```
ROUTER (config-applet)#event ?

application    Application specific event

cli            CLI event

config        Configuration policy event

counter       Counter event

env           Environmental event

interface     Interface event

ioswdsysmon   IOS WDSysMon event

ipsla        IPSLA Event

nf           NF Event

none         Manually run policy event

oir          OIR event

resource     Resource event

rf           Redundancy Facility event

routing      Routing event
```

rpc	Remote Procedure Call event
snmp	SNMP event
snmp-notification	SNMP Notification Event
syslog	Syslog event
tag	event tag identifier
timer	Timer event
track	Tracking object event

Some other useful events are changes in the routing table, IP SLA, object tracking and configuration changes. There is also a big list of possible actions:

ROUTER (config-applet)#**action 1.0 ?**

add	Add
append	Append to a variable
break	Break out of a conditional loop
cli	Execute a CLI command
cns-event	Send a CNS event
comment	add comment
context	Save or retrieve context information
continue	Continue to next loop iteration
counter	Modify a counter value
decrement	Decrement a variable
divide	Divide
else	else conditional

elseif	elseif conditional
end	end conditional block
exit	Exit from applet run
force-switchover	Force a software switchover
foreach	foreach loop
gets	get line of input from active tty
handle-error	On error action
help	Read/Set parser help buffer
if	if conditional
increment	Increment a variable
info	Obtain system specific information
mail	Send an e-mail
multiply	Multiply
policy	Run a pre-registered policy
publish-event	Publish an application specific event
puts	print data to active tty
regexp	regular expression match
reload	Reload system
set	Set a variable
snmp-trap	Send an SNMP trap
string	string commands
subtract	Subtract

syslog	Log a syslog message
track	Read/Set a tracking object
wait	Wait for a specified amount of time
while	while loop

Running CLI commands and sending e-mails are maybe the most important ones but you can also generate SNMP traps or reload the router.

6. IP SLA and EEM Script

```
ROUTER #show running-config | begin ip sla
```

```
ip sla 1
```

```
icmp-echo 192.168.12.2
```

```
frequency 10
```

```
ip sla schedule 1 life forever start-time now
```

```
ROUTER (config)#track 1 ip sla 1 reachability
```

```
ROUTER (config)#event manager applet TRACK_IP_DOWN
```

```
ROUTER (config-applet)#event track 1 state down
```

```
ROUTER (config-applet)#action 1.0 syslog msg "IP SLA 1 is down"
```

```
R1(config-applet)#action 2.0 mail server "smtp.mailserver.local" to
"sagardhawan@networkjourney.com" from "sagardhawan@networkjourney.com" subject "IP
SLA 1 is down" body "IP SLA 1 is not receiving ICMP echo replies anymore"
```

As soon as the object goes down, EEM will perform two actions:

- We produce a syslog message which says "IP SLA 1 is down".
- We send an e-mail to e-mail server "smtp.mailserver.local" using the email addresses, subject and body that I specified above.

We'll also configure an action that will be performed when the object is up again:

```
ROUTER (config)#event manager applet IP_SLA_1_UP  
  
ROUTER (config-applet)#event track 1 state up  
  
ROUTER (config-applet)#action 1.0 syslog msg "IP SLA 1 is up"
```

Once the object is up, we will generate a syslog message. Let's verify our work...

Verifications:

```
R1#show ip sla statistics  
  
IPSLAs Latest Operation Statistics  
  
IPSLA operation id: 1  
  
    Latest RTT: 3 milliseconds  
  
Latest operation start time: 10:16:41 UTC Thu Feb 18 2016  
  
Latest operation return code: OK  
  
Number of successes: 56  
  
Number of failures: 0  
  
Operation time to live: Forever
```

Now we will shut the interface on R2:

```
R2(config)#interface GigabitEthernet 0/1
```

```
R2(config-if)#shutdown
```

Here's what happens on R1:

```
ROUTER #
```

```
%TRACK-6-STATE: 1 ip sla 1 reachability Up -> Down
```

```
%HA_EM-6-LOG: IP_SLA_1_DOWN: IP SLA 1 is down
```

The first message is produced by object tracking. It notices that IP SLA has reported a failure. The second message is produced by EEM and it's the first action that we configured, the syslog message.

Here's the second EEM action:

```
ROUTER #
```

```
%HA_EM-6-LOG: fh_send_mail: : DEBUG(smtp_lib) : &lt;?xml version="1.0" encoding="UTF-8"
?&gt;&lt;fh_smtp_args&gt;&lt;fh_smtp_port&gt;25&lt;/fh_smtp_port&gt;&lt;fh_smtp_secure&gt;
;0&lt;/fh_smtp_secure&gt;&lt;/fh_smtp_args&gt;
```

```
%HA_EM-6-LOG: IP_SLA_1_DOWN : DEBUG(smtp_lib) : smtp_connect_attempt: 1
```

PYTHON 3.X

PYTHON THEORY BASIC TO ADVANCE



PYTHON3
THEORY.pdf

PYTHON NETWORK MODULES:

1. Telnetlib
2. Paramiko
3. Netmiko
4. Napalm
5. Pyntc
6. Nornir
7. Subprocess
8. Scappy

TELNETLIB #1

```
import telnetlib
```

```
HOST = "172.16.221.106"
```

```
user = "admin"
```

```
password = "cisco"
```

```
tn = telnetlib.Telnet(HOST) # to open telnet connection
```

```
tn.read_until(b"Username: ") # Read until a given string, expected, is encountered or until  
timeout seconds have passed.
```

```
tn.write(user.encode(utf8) + b"\n") # always encode from string to bytes when through  
connection
```

```
if password:
```

```
    tn.read_until(b"Password: ") # Read until a given string, expected, is encountered or until  
    timeout seconds have passed.
```

```
    tn.write(password.encode(utf8) + b"\n") # we are connected to network device
```

```
tn.write(b"term len 0\n") # \n represents end of line
```

```
tn.write(b"sh run\n")
```

```
print(tn.read_all().decode(utf8)) # Read all data until EOF; block until connection closed.
```

TELNETLIB #2

```
import getpass
```

```
import telnetlib
```

```

HOST = "172.16.221.106"
user = "admin"
password = "cisco"

tn = telnetlib.Telnet(HOST) # to open telnet connection

tn.read_until(b"Username: ") # Read until a given string, expected, is encountered or until
timeout seconds have passed.
tn.write(user.encode('utf8') + b"\n") # always encode from string to bytes when through
connection
if password:
    tn.read_until(b"Password: ") # Read until a given string, expected, is encountered or until
timeout seconds have passed.
    tn.write(password.encode('utf8') + b"\n") # we are connected to network device

tn.write(b"enable\n") # \n represents end of line
tn.write(b"cisco\n")
tn.write(b"term len 0\n")
tn.write(b"sh run\n")
tn.write(b"conf t\n")
tn.write(b"int loop 0\n")
tn.write(b"ip address 1.1.1.1 255.255.255.255\n")
tn.write(b"int loop 1\n")
tn.write(b"ip address 2.2.2.2 255.255.255.255\n")
tn.write(b"router ospf 1\n")
tn.write(b"network 0.0.0.0 255.255.255.255 area 0\n")
tn.write(b"end\n")
tn.write(b"exit\n") # to close the connection

print(tn.read_all().decode('utf8')) # Read all data until end of file <EOF>; block until connection
closed

```

NETMIKO #1

```

from netmiko import Netmiko

net_connect = Netmiko(host="192.168.32.200", username="admin", password="cisco",
device_type="cisco_ios")

output = net_connect.send_command("show ip int brief")
print(output)

net_connect.disconnect()

```

NETMIKO #2

```

from netmiko import Netmiko
from netmiko import ConnectHandler #connecthandler = functions

#connection = Netmiko(host='10.1.1.1', username='admin', password='cisco',
device_type='cisco_ios')
cisco_device = { #cisco_devices = called as classes
    'device_type': 'cisco_ios',
    'ip': '192.168.32.200',
    'username': 'admin',
    'password': 'cisco',
    'port': 22,
    'secret': 'cisco',
    'verbose': True
}

connection = ConnectHandler(**cisco_device) #connection = object, ** = dictionary used to call
function
output = connection.send_command('show run')
print(output)

```

NAPALM #1

```

from napalm import get_network_driver
import json

driver123 = get_network_driver('ios')

optional_args123 = {'secret': 'cisco'}
ios123 = driver123('192.168.32.200', 'admin', 'cisco', optional_args=optional_args123)
ios123.open()

#start your code here
output123 = ios123.get_arp_table()
for result123 in output123:
    print(result123)
#stop your code here

#dump123 = json.dumps(output123, sort_keys=True, indent=4) #arguments
#print(dump123)

with open('arp.txt', 'w') as f:
    #f.write(dump123)

```

```
ios123.close()
```

ANSIBLE

There are 3 main files in the Ansible directory,

1. hosts,
2. ansible.cfg, and
3. Ansible Playbook file.

Ansible Installation on Linux/Ubuntu:

```
sudo apt install python-pip
```

```
sudo apt install ansible
```

ANSIBLE #1

```
root@ubuntu:/etc/ansible# nano hosts
[AUTOMATION-SWITCHES]
192.168.32.200 ansible_ssh_user=admin ansible_ssh_pass=cisco
#[AUTOMATION-SWITCHES:vars]
#ansible_network_os=ios
```

Create 1ansible.yml

```
---
- hosts: AUTOMATION-SWITCHES
  gather_facts: false
  connection: local
  gather_facts: network_cli

  tasks:
    - name: show run
      ios_command:
        commands:
          - show ip int br
```

ANSIBLE #2

```
---
- name: Running show commands on Cisco IOS
  hosts: AUTOMATION-SWITCHES
```

```
gather_facts: false
connection: network_cli

tasks:
  - name: Run multiple commands on Cisco IOS nodes
    ios_command: #ansible module
      commands: #commands to run
        - show version
        - show ip interface brief

    register: output #register the output in a variable named output

  - debug: var=output.stdout_lines #print the variable at the console line by line
```