

## Table of Contents

<b>Lab 01: Cisco DNA Center Platform – Authentication .....</b>	<b>2</b>
<b>Cisco DNA Center Platform - Authentication .....</b>	<b>2</b>
Objectives.....	2
Prerequisites.....	2
DevNet Sandbox Details .....	3
Authentication API by Authorization string .....	3
Authentication API by Username, password .....	4
<b>Lab 02: Cisco DNA Center Platform - Network Devices .....</b>	<b>6</b>
<b>Cisco DNA Center - Network Devices .....</b>	<b>6</b>
Objectives.....	6
Prerequisites.....	6
<b>Retrieving a list of network devices with Postman .....</b>	<b>6</b>
Checking specific devices.....	7
<b>Creating a network device list with a Python function .....</b>	<b>8</b>
Complete Code .....	10
<b>Retrieving device interface information with Postman .....</b>	<b>10</b>
<b>Creating a list of network device interfaces with a Python function .....</b>	<b>11</b>
Complete Code .....	12

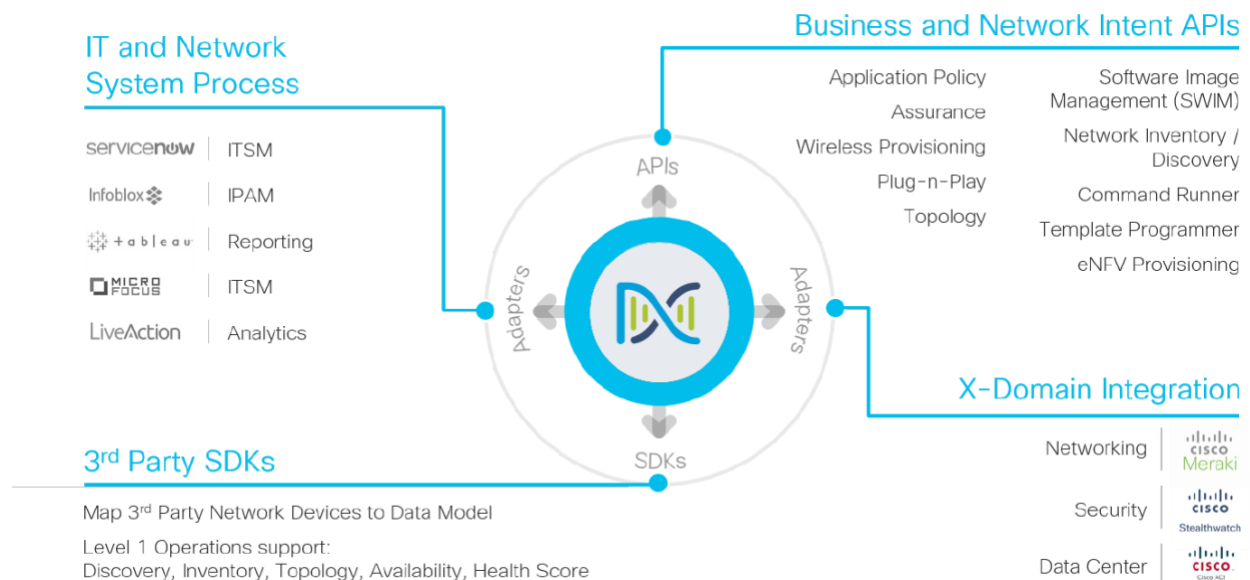
# Lab 01: Cisco DNA Center Platform – Authentication

<https://developer.cisco.com/learning/tracks/programming-dna/dnac-rest-apis/dnac-101-auth/step/1>

## Cisco DNA Center Platform - Authentication

Cisco DNA Center is at the heart of Cisco's new era of networking, the intent-based network.

Cisco DNA Center supports the expression of business intent for network use cases, including base automation capabilities in an enterprise network. The Analytics and Assurance features of Cisco DNA Center provide full context, end-to-end visibility into the network through data and insights.



## Objectives

When you have completed this lab, you will be able to:

- Authenticate and retrieve a token from Cisco DNA Center.
- Build an authentication python function.

## Prerequisites

- [Install Postman](#)
- [Setup Python](#)

## DevNet Sandbox Details

- **Url:** <https://sandboxdnac.cisco.com/>
- **Username:** devnetuser
- **Password:** Cisco123!

## Authentication API by Authorization string

Cisco DNA Center APIs use token-based authentication and HTTPS Basic Authentication to generate an authentication cookie and security token that is used to authorize subsequent requests.

HTTPS Basic uses Transport Layer Security (TLS) to encrypt the connection and data in an HTTP Basic Authentication transaction.

Follow this procedure to set up the request in Postman:

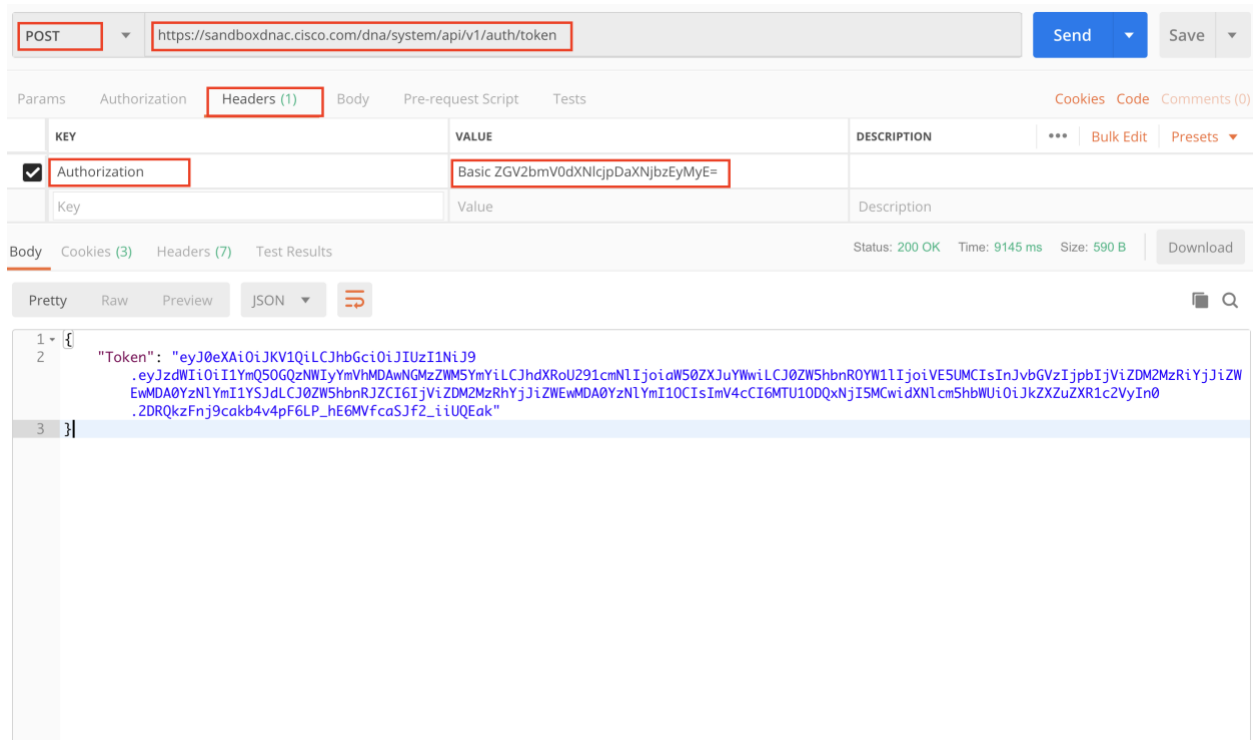
1. Launch Postman and locate the center pane. Here you have the option to select a method for your request. Select **POST** from the drop-down list.
2. In the Request URL section, enter:

**<https://sandboxdnac.cisco.com/dna/system/api/v1/auth/token>**

3. On the Headers tab, create an Authorization key with the value **Basic ZGV2bmV0dXNlcjpDaXNjbzEyMyE=**

**Note:** the authorization value is a Basic Auth Base64 encoding of the username and password provided in the DevNet Sandbox Details section.

4. Click **Send**.



You have successfully made your first Cisco DNAC API Call!

You've just requested a new Token. To authorize subsequent requests, pass the token as the value of the **X-Auth-Token** header of the request. Keep this handy! Later in this learning lab, you'll use this token to send several requests.

## Authentication API by Username, password

Cisco DNA Center APIs use token-based authentication and HTTPS Basic Authentication to generate an authentication cookie and security token that is used to authorize subsequent requests.

HTTPS Basic uses Transport Layer Security (TLS) to encrypt the connection and data in an HTTP Basic Authentication transaction.

Follow this procedure to set up the request in Postman:

1. Launch Postman and locate the center pane. Here you have the option to select a method for your request. Select **POST** from the drop-down list.
2. In the Request URL section, enter:

**https://sandboxdnac.cisco.com/dna/system/api/v1/auth/token**

3. On the Headers Authorization tab, choose Basic Auth and fill in username, password

POST

https://sandboxdnac.cisco.com:443/dna/system/api/v1/auth/token

Send

Save

Params

Authorization

Headers (9)

Body

Pre-request Script

Tests

Settings

Cookies

Code

TYPE

Basic Auth

The authorization header will be automatically generated when you send the request. [Learn more about authorization](#)

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. [Learn more about variables](#)

Username

devnetuser

Password

Cisco123!

☒ Show Password

Body

Cookies

Headers (14)

Test Results

Status: 200 OK Time: 1611 ms Size: 1.19 KB Save Response

Pretty

Raw

Preview

Visualize

JSON

```
1 {  
2   "Token": "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiI2MDJjMGUyODE0NzEwYTYyZDFmNmNlIiwiaWF0IjoiMTYxMjA0ODQ0LWV1Cj00ZShbnR0YW1  
3 }
```

You've just requested a new Token. To authorize subsequent requests, pass the token as the value of the **X-Auth-Token** header of the request. Keep this handy! Later in this learning lab, you'll use this token to send several requests.

# Lab 02: Cisco DNA Center Platform - Network Devices

[https://developer.cisco.com/learning/tracks/programming-dna/dnac-rest-apis/dnac-101-network\\_device/step/1](https://developer.cisco.com/learning/tracks/programming-dna/dnac-rest-apis/dnac-101-network_device/step/1)

## Cisco DNA Center - Network Devices

**Know Your Network** REST request paths can retrieve details about clients, sites, topology and devices, add devices to the network and export device data.

### Objectives

When you have completed this Learning Lab, you will be able to:

- Pull network device list from your network.
- Build a Python function and parse the data.

### Prerequisites

- [Install Postman](#)
- [Setup Python](#)

### Retrieving a list of network devices with Postman

The Network Device Detail Intent API retrieves detailed information about devices by timestamp, MAC address, UUID, name, or nwDeviceName.

Additional REST request paths allow you to retrieve additional information, such as functional capabilities, interfaces, device config, certificate validation status, values of specified fields, modules, and VLAN data associated with specified interfaces. You can also add, delete, update or sync specified devices.

Let's look at how we can pull a network device list that we can tie to an asset management system.

<div> <div> <b>DNA</b> CENTER         </div> <div>           DESIGN           POLICY           <b>PROVISION</b> ASSURANCE           PLATFORM         </div> </div>																																																																													
<div> <div>Devices</div> <div>Fabric</div> </div>																																																																													
<div>Device Inventory</div>																																																																													
<div> <div>Inventory (4)</div> <div>Unclaimed Devices</div> </div>																																																																													
<div>Select device(s) to assign to a Site and Provision network settings from the Network Hierarchy.</div>																																																																													
<div> <div>Last updated: 5:44 pm</div> <div>Refresh</div> <div>Network Telemetry</div> <div>Upgrade Readiness</div> <div>Update Status</div> </div>																																																																													
<div> <div>Filter</div> <div>Actions</div> <div>Tag Device</div> <div>LAN Automation</div> </div>																																																																													
<table> <tr> <th></th><th>Device Name</th><th>Device Family</th><th>IP Address</th><th>Site</th><th>Serial Number</th><th>Uptime</th><th>OS Version</th><th>OS Image</th><th>Last Sync Status</th><th>Credential Status</th><th>Last Provisioned Time</th><th>Provision Status</th></tr> <tr> <td><input type="checkbox"/></td><td>asr1001-x.test1.com</td><td>Routers</td><td>10.10.22.74</td><td>...MiddleEast/HQ</td><td>FXS1932Q1SE</td><td>19 days, 12:58:30.91</td><td>16.3.2</td><td>asr1001e-unik-Tag Golden</td><td>Managed</td><td>Not Provisioned</td><td>May 23 2019 00:23:31</td><td>Failed</td></tr> <tr> <td><input type="checkbox"/></td><td>cat_9k_2.test1.com</td><td>Switches and Hubs</td><td>10.10.22.70</td><td>...MiddleEast/HQ</td><td>FCW2140L039</td><td>19 days, 12:08:48.79</td><td>16.6.4a</td><td>packages.conf Tag Golden</td><td>Managed</td><td>Not Provisioned</td><td>May 23 2019 04:52:27</td><td>Success</td></tr> <tr> <td><input type="checkbox"/></td><td>New.test1.com</td><td>Switches and Hubs</td><td>10.10.22.66</td><td>...I/Sydney/SYD1</td><td>FCW2136L0AK</td><td>19 days, 11:59:50.30</td><td>16.6.1</td><td>packages.conf Tag Golden</td><td>Managed</td><td>Not Provisioned</td><td>May 23 2019 00:23:41</td><td>Failed</td></tr> <tr> <td><input type="checkbox"/></td><td>Test06.test1.com</td><td>Switches and Hubs</td><td>10.10.22.73</td><td>...I/Sydney/SYD1</td><td>FOC1833X0AR</td><td>19 days, 13:49:57.67</td><td>16.6.2s</td><td>packages.conf Tag Golden</td><td>Managed</td><td>Not Provisioned</td><td>May 23 2019 04:52:36</td><td>Success</td></tr> </table>														Device Name	Device Family	IP Address	Site	Serial Number	Uptime	OS Version	OS Image	Last Sync Status	Credential Status	Last Provisioned Time	Provision Status	<input type="checkbox"/>	asr1001-x.test1.com	Routers	10.10.22.74	...MiddleEast/HQ	FXS1932Q1SE	19 days, 12:58:30.91	16.3.2	asr1001e-unik-Tag Golden	Managed	Not Provisioned	May 23 2019 00:23:31	Failed	<input type="checkbox"/>	cat_9k_2.test1.com	Switches and Hubs	10.10.22.70	...MiddleEast/HQ	FCW2140L039	19 days, 12:08:48.79	16.6.4a	packages.conf Tag Golden	Managed	Not Provisioned	May 23 2019 04:52:27	Success	<input type="checkbox"/>	New.test1.com	Switches and Hubs	10.10.22.66	...I/Sydney/SYD1	FCW2136L0AK	19 days, 11:59:50.30	16.6.1	packages.conf Tag Golden	Managed	Not Provisioned	May 23 2019 00:23:41	Failed	<input type="checkbox"/>	Test06.test1.com	Switches and Hubs	10.10.22.73	...I/Sydney/SYD1	FOC1833X0AR	19 days, 13:49:57.67	16.6.2s	packages.conf Tag Golden	Managed	Not Provisioned	May 23 2019 04:52:36	Success
	Device Name	Device Family	IP Address	Site	Serial Number	Uptime	OS Version	OS Image	Last Sync Status	Credential Status	Last Provisioned Time	Provision Status																																																																	
<input type="checkbox"/>	asr1001-x.test1.com	Routers	10.10.22.74	...MiddleEast/HQ	FXS1932Q1SE	19 days, 12:58:30.91	16.3.2	asr1001e-unik-Tag Golden	Managed	Not Provisioned	May 23 2019 00:23:31	Failed																																																																	
<input type="checkbox"/>	cat_9k_2.test1.com	Switches and Hubs	10.10.22.70	...MiddleEast/HQ	FCW2140L039	19 days, 12:08:48.79	16.6.4a	packages.conf Tag Golden	Managed	Not Provisioned	May 23 2019 04:52:27	Success																																																																	
<input type="checkbox"/>	New.test1.com	Switches and Hubs	10.10.22.66	...I/Sydney/SYD1	FCW2136L0AK	19 days, 11:59:50.30	16.6.1	packages.conf Tag Golden	Managed	Not Provisioned	May 23 2019 00:23:41	Failed																																																																	
<input type="checkbox"/>	Test06.test1.com	Switches and Hubs	10.10.22.73	...I/Sydney/SYD1	FOC1833X0AR	19 days, 13:49:57.67	16.6.2s	packages.conf Tag Golden	Managed	Not Provisioned	May 23 2019 04:52:36	Success																																																																	

1. Launch Postman and locate the center pane. Here you have the option to select a method for your request. Select **GET** from the drop-down list.
2. In the Request URL section, enter <https://sandboxdnac.cisco.com/api/v1/network-device>.
3. On the Headers tab, create an **x-auth-token** key. For the value, use the token that you generated in the Learning Lab **Cisco DNA Center Platform - Authentication**.
4. Click **Send**.

GET

https://sandboxdnac.cisco.com/api/v1/network-device

Send

Save

Params

Authorization

Headers (1)

Body

Pre-request Script

Tests

Cookies

Code

Comments (0)

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> x-auth-token	eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWUiOiIYmQ5OGQzNkYyYmVhMDAwNGMzZWMyYmYiLCJhdXRoU291cmNlIjoiaW50ZXJyYjZlZWwMDA0YzNiYmI1YjZlQjZlZWwMDA0YzNiYjZlZWwMDA0YzNiYmI1OCIsImV4cCI6MTU1ODY3MTE1MSwidXN1cm5hbWUiOiJkZXZuZXRxR1c2Vyln0iL0XipG4Nor2f67uZ6csjKcy2GK5Ka5xRakjly0zNF9Y	Description

Body

Cookies (3)

Headers (9)

Test Results

Status: 200 OK

Time: 285 ms

Size: 5.58 KB

Download

Pretty

Raw

Preview

JSON

```

1 {
2   "response": [
3     {
4       "family": "Routers",
5       "errorCode": null,
6       "type": "Cisco ASR 1001-X Router",
7       "lastUpdated": "2019-05-24 03:35:08",
8       "tagCount": "0",
9       "location": null,
10      "role": "BORDER ROUTER",
11      "inventoryStatusDetail": "<status><general code=\\\"FAILED_FEAT\\\"/><topCause code=\\\"UNKNOWN\\\"/>\n</status>",
12      "lastUpdateTime": "1558668908592",
13      "errorDescription": null,
14      "macAddress": "00:c8:8b:80:bb:00",
15      "hostname": "asr1001-x.test1.com",
16      "serialNumber": "FXS1932Q1SE",
17      "softwareVersion": "16.3.2",
18      "locationName": null,
19      "uptime": "15 days, 15:52:24.82",
20      "softwareType": "IOS-XE",
21      "collectionInterval": "Global Default",
22      "roleSource": "MANUAL",
23      "apManagerInterfaceIp": "",

```

## Checking specific devices

If you know the IP address or MAC address of a device and only need to check that specific device, you can use the query parameters `managementIpAddress` or `macAddress`, with the appropriate value for the device.

The screenshot shows a REST client interface with a GET request to `https://sandboxdance.cisco.com/api/v1/network-device?managementIpAddress=10.10.22.74&macAddress=...`. The query parameters are listed in a table:

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> managementIpAddress	10.10.22.74	
<input checked="" type="checkbox"/> macAddress	00:c8:8b:80:bb:00	

The response body is a JSON object:

```
{
  "macAddress": "00:c8:8b:80:bb:00",
  "hostname": "asr1001-x.test1.com",
  "serialNumber": "FXS1932Q1SE",
  "softwareVersion": "16.3.2",
  "locationName": null,
  "upTime": "15 days, 16:43:24.56",
  "softwareType": "IOS-XE",
  "collectionInterval": "Global Default",
  "roleSource": "MANUAL",
  "apManagerInterfaceIp": "",
  "associatedWlcIp": "",
  "bootDateTime": "2019-05-08 11:43:02",
  "collectionStatus": "Managed",
  "interfaceCount": "12",
  "lineCardCount": "9",
  "lineCardId": "cc63ea39-baba-4a54-a109-03b30692c553, faf175f3-7c32-4eda-bdd8-d08fe31be708, 2e0b165f-947e-4ba3-94d4-593583f8100f, 0012fd6e-e048-44b3-9f59-ebbf5937748a, 37831b61-6bb9-4f70-8638-cc4e027a78a7, 8b9d7b61-8d00-43eb-90d4-5e07f1c965da, b5de9856-65f3-4519-adc1-375503774ff9, 124e0995-52e0-4513-9285-6bbd5d995f4e, fa4a05d9-f7c7-40e7-8549-128d7fb479b3",
  "managementIpAddress": "10.10.22.74"
}
```

## Next steps

You have successfully pulled the device list. but this is static, and we need to programmatically pull the list of network devices the controller manages!

## Creating a network device list with a Python function

Let's use the Python `requests` library to create a function that when called upon, will return and display the list of devices managed by the controller.

1. The first part of the function will import the required libraries.
  - **requests** is the library of choice to make the api request.
  - **HTTPBasicAuth** is part of the **requests** library and is used to encode the credentials to Cisco DNA-C
  - **dnac\_config** is a Python file that contains Cisco DNA Center configuration info. In this case we are using our DevNet [Sandbox](#)



2. **import** requests
3. **from** requests.auth **import** HTTPBasicAuth
4. **from** dnac\_config **import** DNAC, DNAC\_PORT, DNAC\_USER, DNAC\_PASSWORD

5. You will now define the function and write the GET request.

6. token = get\_auth\_token() # Get a Token
7. url = "https://sandboxdnac.cisco.com/api/v1/network-device" #Network Device endpoint
8. hdr = {'x-auth-token': token, 'content-type': 'application/json'} #Build header Info
9. resp = requests.get(url, headers=hdr) # Make the Get Request
10. device\_list = resp.json() #capture the data from the controller
11. print\_device\_list(device\_list) #pretty print the data we want

**Note** `print_device_list()` and `get_auth_token()` functions are included part of the complete code.

12. Apply a filter to the data and look for a specific device by creating a query string variable `queryString` and passing the variable part of the `params` parameter in your `requests.get` call.

13. token = get\_auth\_token() # Get a Token
14. url = "https://sandboxdnac.cisco.com/api/v1/network-device" #Network Device endpoint
15. hdr = {'x-auth-token': token, 'content-type': 'application/json'} #Build header Info
16. querystring = {"macAddress": "00:c8:8b:80:bb:00", "managementIpAddress": "10.10.22.74"}
17. resp = requests.get(url, headers=hdr, params=querystring) # Make the Get Request
18. device\_list = resp.json() # Capture data from the controller
19. print\_device\_list(device\_list) # Pretty print the data

20. Execute the code.

21. **if** \_\_name\_\_ == "\_\_main\_\_":
22.   get\_device\_list()

If the code is correct, it will generate output similar to the following.

hostname ptime	mgmt IP	serial	platformId	SW Version	role	U
asr1001-x.test1.com ORDER ROUTER	10.10.22.74 16 days, 16:57:20.36	FXS1932Q1SE	ASR1001-X	16.3.2	B	
cat_9k_2.test1.com CCESS	10.10.22.70 16 days, 16:10:33.49	FCW2140L039	C9300-24UX	16.6.4a	A	
New.test1.com CCESS	10.10.22.66 16 days, 16:17:26.27	FCW2136L0AK	C9300-24UX	16.6.1	A	
Test06.test1.com DISTRIBUTION	10.10.22.73 16 days, 18:08:47.29	FOC1833X0AR	WS-C3850-48U-E	16.6.2s		

Congratulations! You have successfully pulled a list of network devices (or a specific subset of devices) and you are ready to integrate the information into the asset management tool of your choice.

## Complete Code

Click below to view the [complete script](#)

## Retrieving device interface information with Postman

Now that we know the list of devices the controller is managing, let's pull the device interface information. This can be used to audit your network, automate a task to apply port security and create policies based on usage.

1. Launch Postman and locate the center pane. Here you have the option to select a method for your request. Select **GET** from the drop-down list.
2. In the Request URL section, enter <https://sandboxdnac.cisco.com/api/v1/interface>.
3. On the Headers tab, create an **x-auth-token** key. For the value, use the token that you generated in the Learning Lab **Cisco DNA Center Platform - Authentication**.
4. On the Params tab, create a **deviceId** key. For the value, use a device ID that you pulled in the previous section of this Learning Lab.

GET  Send Save

Params Authorization Headers (1) Body Pre-request Script Tests Cookies Code Comments (0)

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
<input checked="" type="checkbox"/> deviceId	64b415be-502a-419d-b57a-c0390c03b2ed			
<input type="checkbox"/> status	up			
Key	Value	Description		

Body Cookies (3) Headers (9) Test Results Status: 200 OK Time: 1130 ms Size: 131.92 KB Save Download

Pretty Raw Preview JSON

```

1 {
2   "response": [
3     {
4       "description": " P2P link 3850",
5       "status": "up",
6       "lastUpdated": "2019-05-26 05:19:36.275",
7       "interfaceType": "Physical",
8       "className": "EthrntrPrtclEndpntExtndd",
9       "duplex": "FullDuplex",
10      "portMode": "routed",
11      "portType": "Ethernet Port",
12      "macAddress": "00:c8:8b:80:bb:00",
13      "deviceId": "64b415be-502a-419d-b57a-c0390c03b2ed",
14      "speed": "10000000",
15      "adminStatus": "UP",
16      "ifIndex": "2",
17      "vlanId": "0",
18      "portName": "TenGigabitEthernet0/0/1",
19      "mediaType": "unknown",
20      "series": "Cisco ASR 1000 Series Aggregation Services Routers",

```

**Next:** Creating a list of network device interfaces with a Python function

## Creating a list of network device interfaces with a Python function

Let's use the Python *requests* library to create a function that when called upon returns a list of all interfaces to a specific device.

1. The first part of the function will import the required libraries.
  - o **requests** is the library of choice to make the API request.
  - o **dnac\_config** is a Python file that contains Cisco DNA Center configuration info. In this case we are using our DevNet [Sandbox](#)

2. **import requests**

3. **from dnac\_config import DNAC, DNAC\_PORT, DNAC\_USER, DNAC\_PASSWORD**

2. You will now define the function and write the GET request. This builds on the function that you wrote earlier in this Learning Lab for retrieving a list of devices.

3. **def get\_device\_int(device\_id):**

```

4. """
5. Building out function to retrieve device interface. Using requests.get
6. to make a call to the network device Endpoint
7. """
8. url = "https://sandboxdnac.cisco.com/api/v1/interface"
9. hdr = {'x-auth-token': token, 'content-type': 'application/json'}
10. querystring = {"macAddress": device_id} # Dynamically build the query params to get
    device-specific Interface info
11. resp = requests.get(url, headers=hdr, params=querystring) # Make the Get Request
12. interface_info_json = resp.json()
13. print_interface_info(interface_info_json)

```

**Note** `print_interface_info()` function is part of the complete code sample found [here](#). If the code is correct, it will generate output similar to the following.

portName tUpdated	vlanId	portMode	portType	duplex	status	las
TenGigabitEthernet0/0/1 9-05-26 06:36:07.25	0	routed	Ethernet Port	FullDuplex	up	201
GigabitEthernet0/0/4 9-05-26 06:36:07.25	0	routed	Ethernet Port	FullDuplex	down	201
VoIP-Null0 6 06:36:07.25	0	routed	OTHER	None	up	2019-05-2
GigabitEthernet0/0/2 9-05-26 06:36:07.25	0	routed	Ethernet Port	FullDuplex	down	201
GigabitEthernet0/0/3 9-05-26 06:36:07.25	0	routed	Ethernet Port	FullDuplex	down	201

Congratulations! You have successfully pulled a list of network devices and interfaces.

### Complete Code

Click below to view the [complete script](#)

**Congratulations! You have completed Cisco DNA Center Platform - Network Devices.**