2.3.4.1: Lab - Using the APIC-EM Path Trace API-2

This page summarizes the steps in **Lab - Using the APIC-EM Path Trace API**. Click **here (https://288647296.netacad.com/courses/951643/files/82148741/download?wrap=1)** **(https://288647296.netacad.com/courses/951643/files/82148741/download?wrap=1)** to download the lab PDF if you would like a more detailed explanation.

Background / Scenario

The APIC-EM includes a robust Path Trace application that is accessible through the API. In this lab, you will complete a program that does the following:

- Displays lists of all of the hosts and network devices on the APIC-EM network.
- Accepts user input for the source and destination devices for the path trace.
- Initiates the path trace.
- Monitors the status of the path trace until it has been completed by the APIC-EM.
- Displays details of the completed path trace to the user.

Required Resources

- Access to an APIC-EM sandbox
- IDLE Python IDE
- Python modules:
  - json
  - requests
  - time, a new module for this lab
  - tabulate
  - functions file previously created in the workshop or the corresponding solution files
- JSON sample data file **path_trace_data.json**

Step 1: Access the APIC-EM API documentation for Flow Analysis.

a. Open the APIC-EM sandbox and access the API documentation page.
b. Open the documentation for the Flow Analysis endpoint. Click **API > Flow Analysis > flow-analysis > POST /flow-analysis.**
c. Under the **Parameters** heading, click **Model Schema** to view the schema for the **/flow-analysis POST** method.
d. Make note of the schema for the request body. You will create a variable to hold dictionary key/value pairs for the source and destination IP addresses only.
e. Under the **Response Class** heading, click **Model Schema** and make note or the key/value pairs in the **response** dictionary. You will parse the returned JSON for the **flowAnalyisId** key and use its value to monitor the progress of the path trace.

Step 2: Complete the code for Section 1 to setup the environment.

a. Open the **04_path_trace.py** work file in IDLE.
b. In **Section 1**, enter the code to import the required modules using the commands you learned in previous labs. The list of required modules appears in the **Required Resources** section above.
c. Disable SSL certificate warnings. **Hint**: This code was used in **get_ticket.py**.
d. Create the **api_url** variable and assign it the URL of the Flow Analysis endpoint of the API. The URL is **https://{YOUR-APICEM}.cisco.com/api/v1/flow-analysis**.
e. Add code to get a fresh service ticket using the function you created earlier. Assign the returned value to the variable **ticket** to be used in the POST request headers.
f. Create a dictionary called **headers** to hold the HTTP header information that will be supplied to the POST request, as was done in previous labs in **print_hosts.py** and **print_devices.py**.

Step 3: Complete the code for Section 2 to display the hosts and network devices inventory.

Refer to **Section 2** of the code. In this step, you will reuse the functions that you created previously to display lists of the available hosts and network devices that may be included as endpoints in a Path Trace.

a. Print a message that tells the user what is being displayed followed by the function that displays the list of **hosts** in the topology on a new line. Use the following syntax:

```
print("string")
```

```
function_name()
```

b. Print another message that tells the user what is being displayed followed by the function that displays the list of **network devices** in the topology.
c. To test your code, copy Sections 1 and 2 into a new IDLE document, save the file, and then run the code.

Step 4: Complete the code for Section 3 to request the source and destination IP addresses.

Refer to **Section 3** of the code. As you can see from the Path Trace API documentation, requests to the **/flow-analysis** API can accept a number of values. For this lab, you will only be using source and destination IP addresses, but the additional parameters can be added by modifying the JSON that is submitted to the API.

a. Prompt the user for the required IP addresses and assign the input to the **s_ip** (source) and **d_ip** (destination) variables using the following syntax:

```
variable = input("prompt string: ")
```

b. Remember that you are working within a **while** loop structure. Indentation is important. Be sure that your level of indentation is consistent with other code that is present in the **while** loop.
c. **Optional Challenge**: Add two statements that will get the values of the **path_data** dictionary keys and print a message that displays the source and destination IP addresses entered by the user.
d. Copy the code in Section 3 and paste it into your test file. Run, test, and debug the code. Try entering IP addresses at the prompts and try skipping one of the entries by pressing the <Enter> key without entering an address to test your error messages.

Step 5: Complete the code for Section 4 to initiate the path trace and get the flow analysis ID.

Refer to **Section 4** of the code. In order to initiate the Path Trace, the **requests.post()** method is used to submit the source and destination IP addresses to the **/flow-analysis** API. The **request.post()** method works similarly to the other requests module methods. You will pass the method the four parameters with the variables shown in the table below:

Parameter	Variable/	Explanation

	Parameter Name	
URL	api_url	The URL of the API endpoint.
body	path	JSON formatted data for the source and destination IP addresses.
headers	headers	The content type and authentication token represented as JSON data.
verification	verify	Used to verify the server's security (TLS) certificate. False means the certificate is not verified.

a. Create the **path** variable and assign the converted **path_data** dictionary variable to it. The syntax is:

```
variable_json = json.dumps(dictionary_variable)
```

b. Build the **post()** method to submit the path trace request, and store the response in the **resp** variable. Look at the example of the use of the **requests.post()** method from the **get_ticket** code for guidance and use the information in the table above. Supply the correct variables to the statement. The syntax is:

```
response_variable = requests.post(URL,body,headers=variable,verify=False)
```

c. You will need the value of the **flowAnalysisID** to check the status of the path trace request and to get the completed path trace data from the APIC-EM. Look at the structure of the JSON that is returned by the API.

```
{
  "version": "",
  "response": {
    "flowAnalysisId": "",
    "url": "",
    "taskId": ""
  }
}
```

From this you can see that that the JSON exists at two levels. The **version:** and **response:** objects are at the first level. At the second level, you can see the **flowAnalysisId** and other values. Use the **resp_json** variable that has been created in the code to translate the JSON into a Python dictionary from which you can extract the **flowAnalysisID**. The syntax is:

```
flowAnalysisId = resp_json["level one key"]["level 2 key"]
```

d. Run your code and fix any errors that may appear.

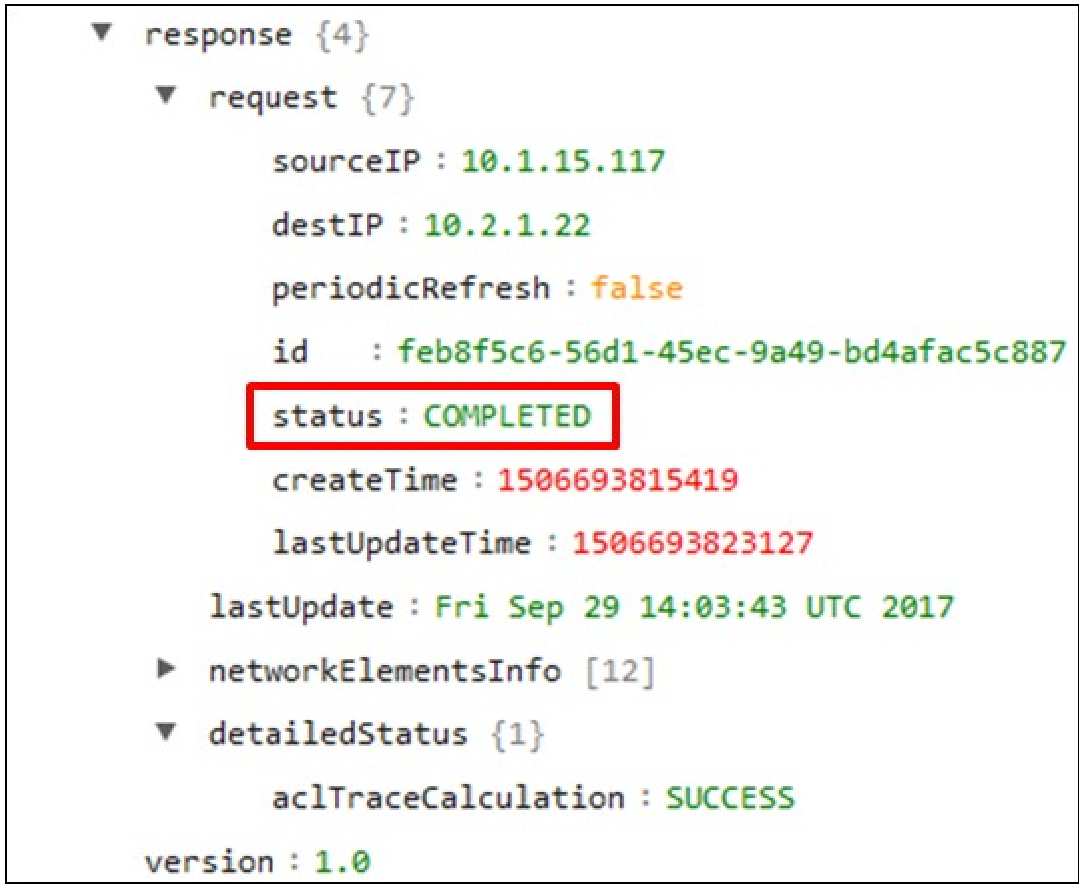
Step 6: Complete the code in Section 5 to check status of the Path Trace request.

Refer to **Section 5** of the code. The Path Trace request that you submitted in the previous step only begins the path trace process and returns the flow analysis ID. To display the results of the Path Trace, you have to wait for the process to complete. To do this, you create a **while** loop that repeatedly request the status of the trace. The status of the request can be COMPLETED, FAILED, or INPROGRESS. If the status is COMPLETED, the program moves on to the next section of code, which will display the results of the Path Trace. If it is FAILED, a message is displayed and the program terminates. Additional code is also included in the while loop to limit the number of iterations and to set a one-second timer.

Note: Not all devices with IP addresses can successfully function as endpoints for a path trace. If your path trace fails, try using only host addresses as endpoints.

a. The URL for requests to the **/flow-analysis** end point can include the flow analysis ID, as shown in the Swagger documentation for the Flow Analysis endpoint. Because multiple path traces could be occurring simultaneously, this allows data retrieval for a specific request. Construct a new variable, called **check_url** that contains the value **api_url** followed by a forward slash and then the flow analysis ID stored in the previous step.

b. The JSON-formatted data that is returned by the Path Trace API contains four or more levels of objects, some with multiple arrays. The status value to determine the status of the request is only three levels in, as shown below:



```
▼ response {4}
    ▼ request {7}
          sourceIP : 10.1.15.117
          destIP : 10.2.1.22
          periodicRefresh : false
          id   : feb8f5c6-56d1-45ec-9a49-bd4afac5c887
          status : COMPLETED
          createTime : 1506693815419
          lastUpdateTime : 1506693823127
      lastUpdate : Fri Sep 29 14:03:43 UTC 2017
    ▶ networkElementsInfo [12]
    ▼ detailedStatus {1}
          aclTraceCalculation : SUCCESS
  version : 1.0
```

The JSON returned is stored in the variable called **response_json**, as shown in the code. You need to extract the value of the status object from **response_json** and put it into the **status** variable. Construct the line of code that will accomplish this task and enter the line into your program for the second instance of **status** inside the while loop.

c. Save and run your code.

Step 7: Complete the code in Section 6 to display the results of the path trace.

Most of the code in Section 6 is already done for you. Read through it to gain an understanding of how the code parses the JSON from the path trace results and then displays it in a table. If you would like a more detailed explanation of how the code works in this section, click here (https://288647296.netacad.com/courses/951643/files/82148741/download?wrap=1) (https://288647296.netacad.com/courses/951643/files/82148741/download?wrap=1) to refer to the lab PDF. The following steps briefly summarize how to complete the variables **path_source**, **path_dest**, and **networkElementsInfo**.

a. Open the **path_trace_data.json** file in a text editor, copy the contents, and paste it into the JSON viewer at https://codebeautify.org/jsonviewer (https://codebeautify.org/jsonviewer) .
b. Generate a tree view of the JSON to determine the keys needed to complete the three variables. Locate the source IP address and destination IP address keys.
c. Complete the **path_source** and **path_dest** variables by providing the correct names for the keys. Use the following syntax:

```
path_source = response_json["level1key"]["level2key"]["level3key"]
path_dest = response_json["level1key"]["level2key"]["level3key"]
```

d. Return to the JSON tree in CodeBeautify.com. Make note of the location of the networkElementsInfo dictionary. Complete the **networkElementsInfo** variable by providing the correct names for the keys. Use the following syntax:

```
networkElementsInfo = response_json["level1key"]["level2key"]
```

Step 8: Save and run the final code file.

Now you should be able to run the entire path trace application. If you have been unable to complete all activities in this workshop, please inspect and run the various solution files that are distributed with this workshop.