Lab – Using the APIC-EM Path Trace API

Objectives

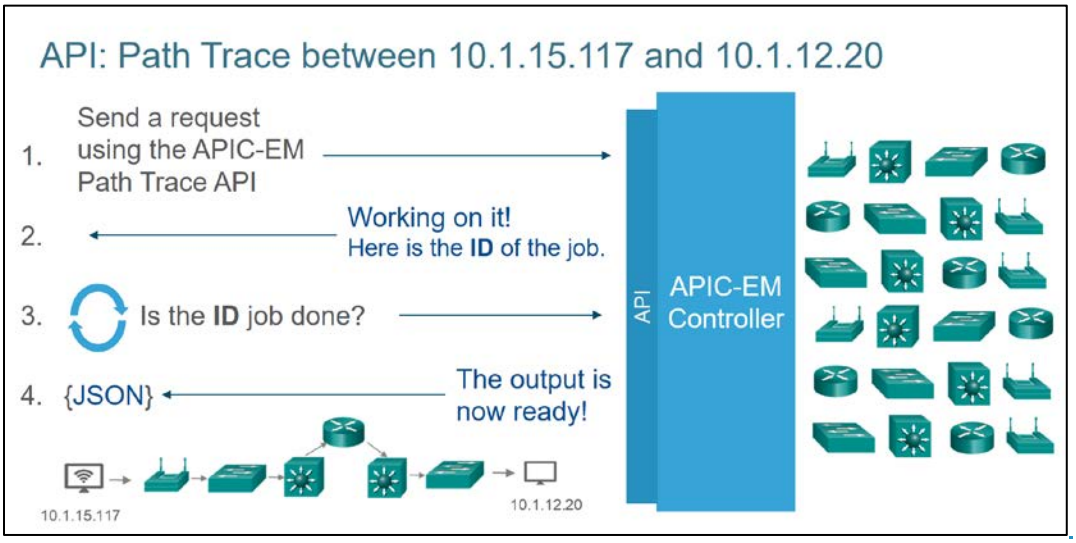
In this lab, you will create a program interacts with APIC-EM Path Trace application.

Background / Scenario

In this lab, you will utilize the APIC-EM API to perform a Path Trace by creating a program in Python, and by using the functions that have been created in the workshop.

The APIC-EM includes a robust Path Trace application that is accessible through the API. 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.



Software developers frequently collaborate with others in the community to share code and access solutions in order to work more efficiently. In this lab, you are encouraged to seek help from others and refer to code that you have already developed. In addition, if necessary, solution code is available that you can explore in order to perform the tasks in this lab.

Note: To achieve the objectives of this workshop, the Python code you will work with has been simplified. This simplification may result in code that is less efficient, and less eloquent, than that created by professional software developers. As you gain skills in software development and network programmability, you will discover techniques that improve the usability of code. You are encouraged to explore the exemplar Path Trace solution that is available for download from the DevNet GitHub repository to experience some of these techniques. This file and other tools can be found here: https://github.com/CiscoDevNet/apic-em-samples-aradford.

© Cisco and/or its affiliates. All rights reserved. Cisco Confidential

Required Resources

- Access to the APIC-EM in the DevNet sandbox at **https://{YOUR-APICEM}.cisco.com**
- 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**

Procedure

You will work to complete a small Python application that communicates with the APIC-EM Path Trace application. You are provided with two Python programs, the file that you will work in and the solution file. Depending on your familiarity with Python, you may want to have both files open. As you work through the lab, you will finish functional blocks of code that work together as the Python application. To run and test your working code, you will copy each completed code block into a new Python file of your own. As the code is completed, you can run the code blocks that you have finished in this file by copying and pasting them. In this way, you will compile the application from the completed code.

Work between the instructions in the lab and the lab code file named **04_path_trace.py**. You will add your own code to the code provided in that file. To locate the places where your input is required, look for comments surrounded by "+" characters, as shown in the example below.

```
#++++++++++++Add Values+++++++++++++++
# Path Trace API URL for flow_analysis endpoint
api_url = !!!REPLACEME!!!    # URL of API endpoint
# Get service ticket number using imported function
ticket = !!!REPLACEME!!!     # Call your function name that returns the service ticket
# Create headers for requests to the API
headers = !!!REPLACEME!!!    # Create dictionary containing headers for the request
#++++++++++++++++++++++++++++++++++++
```

In the **04_path_trace_sol.py** solution file, the areas of completed code are highlighted similarly; however "Add Values" has been removed from the comments because this code is complete.

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 of 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.

© Cisco and/or its affiliates. All rights reserved. Cisco Confidential

www.netacad.com

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. Follow this pattern:

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

b. Print a 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. What should happen?

_____

_____

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** and **d_ip** variables following the pattern below:

```
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.

Look at the **if…else** condition that is used to evaluate the user input. What condition is it testing?

_____

_____

© Cisco and/or its affiliates. All rights reserved. Cisco Confidential

This condition is meant to trap errors in the entries supplied by the user. What other errors could the code detect in the user input to ensure that only valid IP addresses have been entered?

_____

_____

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. What should happen?

_____

_____

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. 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/Parameter Name	Explanation
URL	api_url	The URL of the API endpoint
body	JSON formated path_data	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. In the last step, the source and destination addresses for the Path Trace were requested from the user. The values were stored in a dictionary held by the **path_data** variable. A variable is created from this data. However, the Python dictionary needs to be converted to JSON format that is consumed by the API service. This is done with the **json.dumps()** method, which takes the object to be converted as its argument. In this case, that object is the **path_data** dictionary.

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 **requests.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. Once the Path Trace requests has been submitted, you will need to extract the value of the **flowAnalysisID** from the response. This value will be used to check the status of the Path Trace request

© Cisco and/or its affiliates. All rights reserved. Cisco Confidential

and to get the completed Path Trace data from the APIC-EM. Use the **resp_json** variable that has been created in the code to translate the JSON formatted data from the API into a Python dictionary from which you can extract the **flowAnalysisID** value.

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. Your job is to access the value for the **flowAnalysisId** object and assign it to the variable **flowAnalysisId** in 04_path_trace.py. Write a line of code that follows the pattern below:

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

d. Run your code and fix any errors that may appear. What should be output to the screen at this point in the program?

_____

_____

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 submitted in the previous step only begins the Path Trace process and returns the flow analysis ID. The Path Trace can take time to execute depending on the size of the network and other factors.

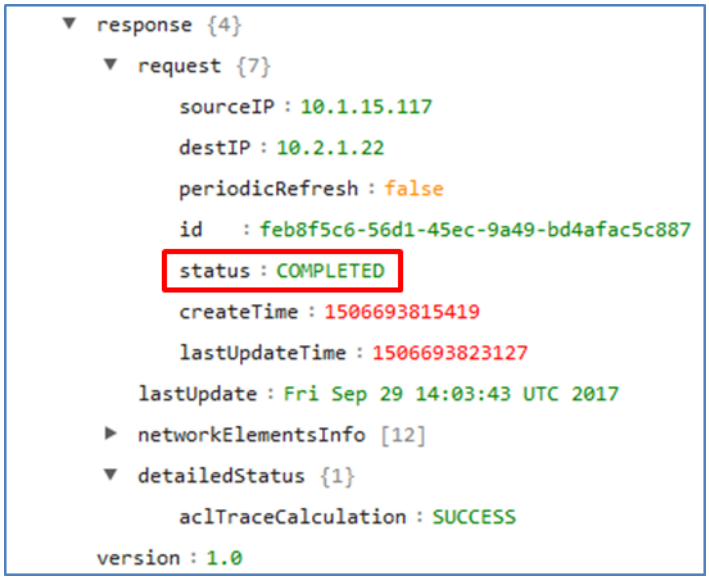
To display the results of the Path Trace, you will wait for the process to complete before you can receive the results. To do this, a **while** loop is established to request the status of the trace repeatedly. The loop will execute until the API returns a status of 'COMPLETED.' The code within the loop repeatedly examines the status key in the data returned by the API, and acts accordingly. 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. In addition, a counter is set that is used to exit the loop if a predetermined number of iterations is exceeded. This is created to avoid an endless loop. Because a one-second timer is set within the loop to give the APIC-EM time to complete the Path Trace, the counter variable roughly corresponds to the amount of time allowed for the loop to complete. If it takes too long for the Path Trace to complete, then an error may exist. The threshold value for the counter can be increased in the code according to network conditions.

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 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 in the **flowAnalysisId** variable.

© Cisco and/or its affiliates. All rights reserved. Cisco Confidential

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



The screen shot shows the high-level structure of the JSON with details of the **request** object expanded. You are interested in the **status** object shown. If you move down the tree from the highest level object, the path to the status object is *response->request->status*. The response is put into a Python dictionary 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 below, and enter the line into your program for the second instance of **status** inside the while loop. Consult previous examples and check with your neighbor if you want to verify your answer first.

status = response_json["_____"]

c. Save and run your code. What do you expect to see displayed on the screen as the code runs?

_____

_____

Step 7: Complete the code in Section 6 to display the results of the Path Trace.

Refer to **Section 6** of the code. At this point, you have successfully posted a Path Trace request by supplying source and destination IP addresses that come from lists of devices that are on the network. However, nothing has yet been displayed regarding the results of the trace itself.

An example of the full JSON is available in a text file called **path_trace_data.json** that is included with the files for this workshop. This JSON data poses several challenges.

- The JSON is long with many nested objects and arrays.
- The information returned for each device on the path varies, with some objects being present for some devices but not for others.

a. Open the **path_trace_data.json** file in a text editor, select and copy the entire contents of the file, and paste it into the JSON viewer at https://codebeautify.org/jsonviewer.

b. Generate a tree view of the JSON and answer the following questions.

© Cisco and/or its affiliates. All rights reserved. Cisco Confidential

1) The top level objects in the JSON are **"response":** and **"version":** (The **object** and **array** levels appear by default in all trees.) What four objects make up the next level of objects under **"response":**?

_____

_____

_____

2) How many objects are there in the array under **"networkElementsInfo":** and what does each object represent?

_____

_____

_____

3) Look at the keys for the devices on the path. What keys are found in some objects but not others?

_____

_____

_____

c. In IDLE, create a new file, and save the file in the folder that contains the files for this workshop. Name the file as you wish. In this file, please do the following:

1) Import the **json** library.

2) Open the **path_trace_data.json** file, convert it to Python objects, and assign the result to a variable called **json_data**. Use the following syntax:

```
json_data = json.load(open("filename.extension"))
```

3) Save and run the program. The program should run and provide you with a Python shell.

d. Display the contents of **json_data** in the shell with **print()**. This is what the imported and converted JSON looks like to Python.

e. Display the contents of the **"request":** object that is under the **"response":** key. Use the following syntax:

```
>>> print(json_data["object1"]["object2"])
```

f. Display the value of the source and destination IP addresses that were posted to the API. This requires references to another level of keys under the **"request"** key. Check the JSON structure in the JSON Viewer for the names of the keys. You will use this code in a program later.

_____

_____

g. The data that is needed to display for the results of the Path Trace is in the **networkElementsInfo** JSON object. This object contains an array, or list of objects, that represent all of the objects on the path from source to destination. You want to extract values from each object and display them to the user after the Path Trace has successfully completed. Because this object is an array, in Python you need to refer to a combination of dictionary keys and list elements to get the information needed for each device.

1) Create a variable called **networkElementsInfo** and assign it the contents of the **networkElementsInfo** object. Display the contents of this variable in the shell.

2) This variable contains a list that contains dictionaries for each device. To verify this, use the Python **len()** function with this variable as its argument to display the number of elements in the list. This should match the number of elements shown in the JSON viewer.

© Cisco and/or its affiliates. All rights reserved. Cisco Confidential

h. Return to your **04_path_trace.py** file. Based on what you did above, supply values for the **path_source**, **path_dest**, and **networkElementsInfo** variables. Leave the shell window open.

i. Return to the shell. You will continue to explore accessing values in the JSON data.

1) You can address slices of the list that represent a single device or multiple devices. Display the information for the first device by displaying the first element of the **devices** variable. Remember that the first element of a list is at position 0. Repeat this for other elements in the list of devices. Follow this pattern:

```
list_variable[# of element]
```

_____

_____

2) Now, address individual keys in the device list elements. This requires you to add a reference to the key for the information that you want to see. Try the "id" key of the first element. Follow this pattern:

```
list_variable[# of element]["key"]
```

_____

_____

Note: Some objects lack some keys like **"name"**, and **"egressInterface",** for example. Other keys are also present for some devices, but not for others.

3) What happens if you attempt to access a key that is not present for the device? For example, try to get the value of the "name" key for the first device. If this code was running in a program, what would happen?

_____

_____

j. Experiment with accessing other keys for the devices. For example, enter an expression that displays the **"name"** value for the **"physicalInterface"** key of the **"ingressInterface"** key of one of the switches or routers in the list of devices.

To do this, inspect the JSON data either in the data file or JSON viewer. Find a switch or router in the list by looking at the "type" attribute. Note the position of the device in the list. Now, access the slice of the device variable for that position. Then, add keys for the nested dictionaries down to the "name" attribute for the physical ingress interface as shown below:

```
>>> networkElementsInfo[9]["ingressInterface"]["physicalInterface"]["name"]
```

k. The last section of the **04_path_trace.py** code creates a list of the information that displays each device present in the Path Trace. This list is passed to the **tabulate()** function, along with a list of the column headings to use for each element in the list. Look at the code and answer the following questions.

1) What values are extracted from the JSON and printed for each device in the path?

_____

_____

2) What is the purpose of the **for… in…** loop?

_____

_____

3) Why are so many **if** conditions required?

_____

_____

© Cisco and/or its affiliates. All rights reserved. Cisco Confidential

www.netacad.com

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.

Challenge

For an additional challenge, try the following activities. Start a GitHub project and share your work with others. Get a few email addresses from other workshop participants, and work together to accomplish these challenge activities.

a. Format the output of the Path Trace for Graphziv. Instead of printing a table, print out a list of devices that is formatted for Graphviz. Run the file and paste the output into the online Graphviz viewer at http://www.webgraphviz.com/. Check the Graphviz documentation for formatting information.

b. Find a partner. Create a new GitHub project and work together, either in class or remotely, to further modularize the code. For example, create generic functions that execute **requests.post()** and **request.get()** methods and return the results of the post to the calling code. This will require passing parameters to the functions. These functions can be saved in the **apic_em_functions.py** file and called from the **04_path_trace.py** file.

c. Create a function that validates IP addresses that have been input by users.

d. Create code that checks if an IP address input by a user is present in the host and device inventory.

© Cisco and/or its affiliates. All rights reserved. Cisco Confidential