Lab - Getting a Service Ticket with Python

Objectives

Part 1: Get a Service Ticket in Postman

Part 2: Get a Service Ticket in Python

Background / Scenario

Before requests can be made to other APIC-EM API endpoints, a service ticket number must be obtained. Each request must include a valid service ticket number that serves as an authentication token. In this lab, you will first use Postman to request a service ticket from the APIC-EM API. You will then duplicate the process of making the request in Python. Finally, you will convert your program into a reusable function that will used in later labs.

This request is addressed to the APIC-EM Role Based Access Control API /ticket endpoint.

Required Resources

- Postman
- Python 3 with IDLE
- Python **requests** module
- Access to the Internet

Note: Use the APIC-EM sandbox URL and credentials provided by your instructor. Only use the public APIC-EM URL and credentials for additional study after the conclusion of the workshop. For example purposes, this activity uses the URL and credentials of the public sandbox.

Part 1: Get a Service Ticket in Postman

In this part, you will request a service ticket number from APIC-EM using Postman.

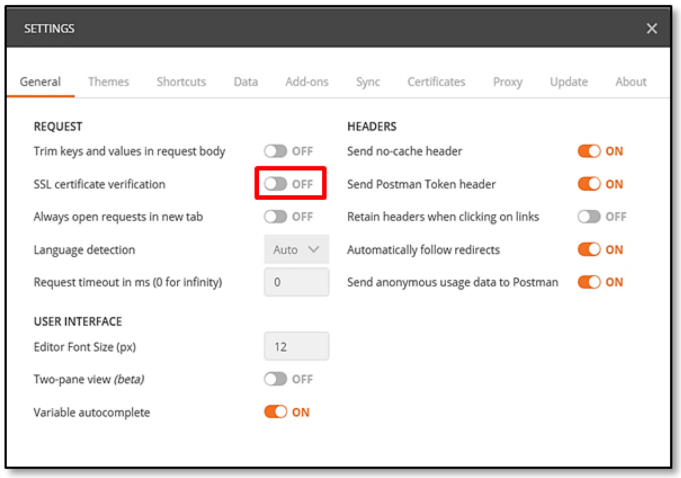
Step 1: Configure Postman.

Configure a Postman setting.

a. Open Postman and create a new tab if necessary.

b. Click **File > Settings**.

c. Under the **General** tab, set the **SSL certificate verification** to **OFF**. Close the **Settings** dialog box.


SETTINGS

General Themes Shortcuts Data Add-ons Sync Certificates Proxy Update About

REQUEST
Trim keys and values in request body OFF
SSL certificate verification OFF
Always open requests in new tab OFF
Language detection Auto
Request timeout in ms (0 for infinity) 0

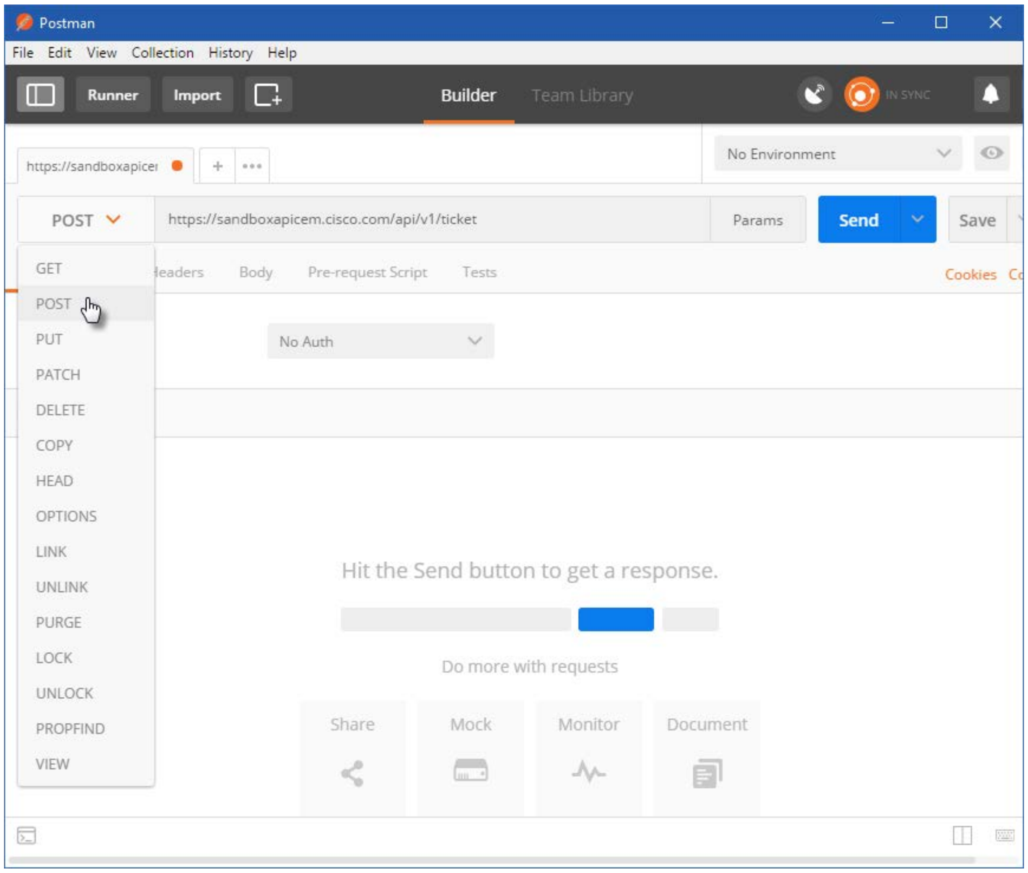
USER INTERFACE
Editor Font Size (px) 12
Two-pane view *(beta)* OFF
Variable autocomplete ON

HEADERS
Send no-cache header ON
Send Postman Token header ON
Retain headers when clicking on links OFF
Automatically follow redirects ON
Send anonymous usage data to Postman ON

Step 2: Select the method and enter the required URL.

a. Next to the URL box, change the request method from **GET** to **POST**.

b. Enter the URL for API endpoint: **https://{YOUR-APICEM}.cisco.com/api/v1/ticket.** (Replace **{YOUR-APICEM}** with the subdomain of the APIC-EM sandbox instance used in this workshop.)

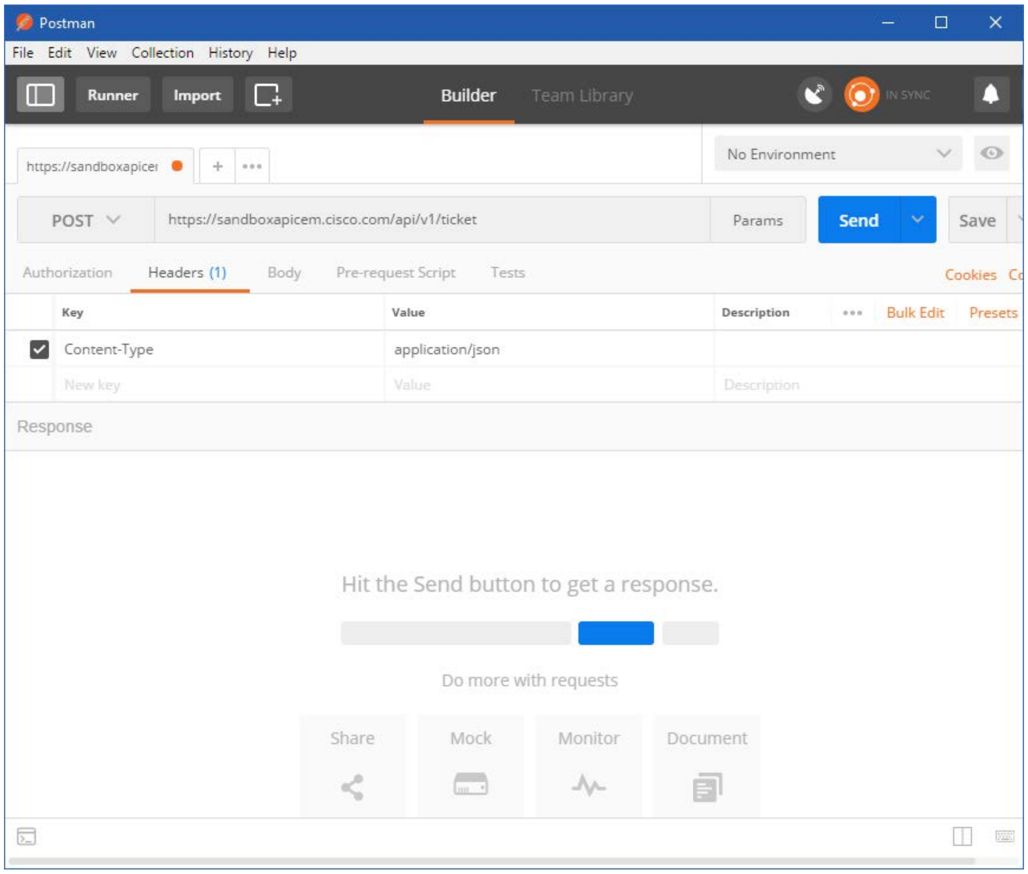

Postman
File Edit View Collection History Help
Runner Import Builder Team Library IN SYNC
https://sandboxapicei
No Environment
POST
https://sandboxapicem.cisco.com/api/v1/ticket
Params Send Save
GET POST PUT PATCH DELETE COPY HEAD OPTIONS LINK UNLINK PURGE LOCK UNLOCK PROPFIND VIEW
Headers Body Pre-request Script Tests Cookies
No Auth
Hit the Send button to get a response.
Do more with requests
Share Mock Monitor Document

Step 3: Enter the header information.

a. Below the URL input field, select **Headers**.

b. Under **Headers**, click the **New key** field under the **Key** column and enter **Content-Type**.

c. In the value column enter **application/json**.



https://sandboxapicem.cisco.com/api/v1/ticket

Key	Value	Description
✓ Content-Type	application/json	
New key	Value	Description

Step 4: Enter the authentication information.

The APIC EM is the **/ticket** endpoint. It requires that authentication information is carried in the body of the request. Use the credentials that have been provided to you by your instructor.

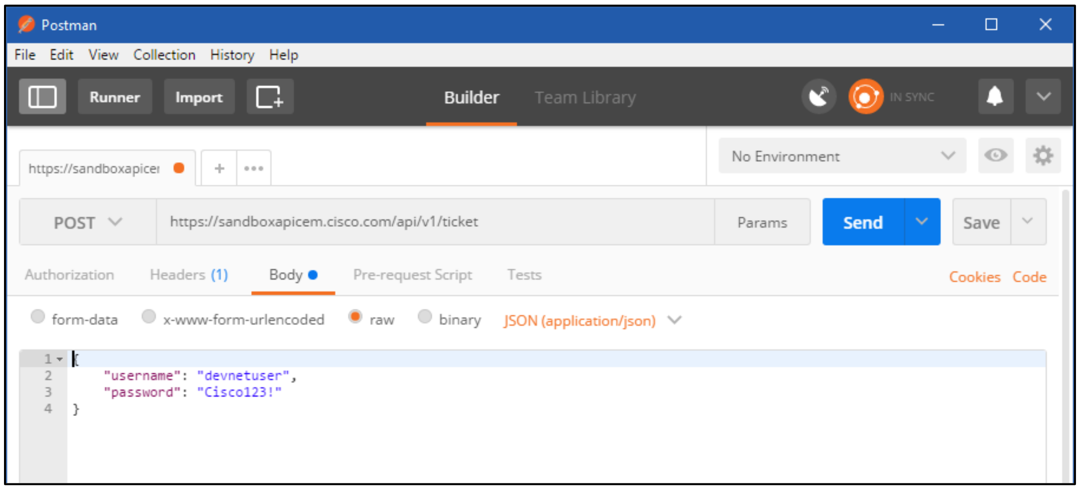
a. Select the **Body** tab that is next to the **Headers** tab.

b. Click **Raw** to format the JSON data.

c. Type the following as it appears below. Replace the values with the appropriate credentials.

```
{
    "username": "!!!REPLACEME with the Username!!!",
    "password": "!!!REPLACEME with the Password!!!"
}
```

Your screen should resemble the one below.



POST https://sandboxapicem.cisco.com/api/v1/ticket

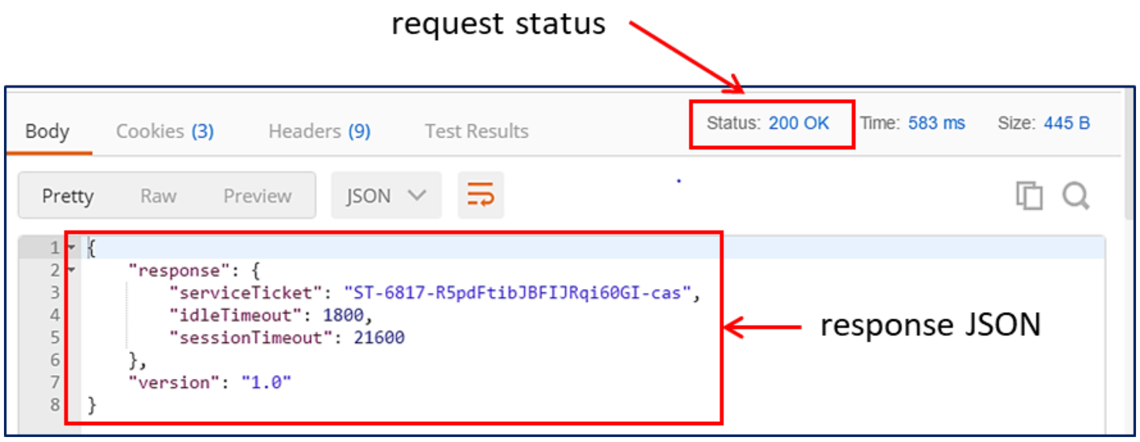
```
{
    "username": "devnetuser",
    "password": "Cisco123!"
}
```

Step 5: Send the request.

a. Click **Send**.

b. After a brief delay, you should see the response JSON appear below the request information. Scroll down, if necessary to view the response data.

c. The response consists of JSON data that includes the service ticket value and some other information about the service ticket. Your **serviceTicket** value will be different than the one shown below.



```
{
    "response": {
        "serviceTicket": "ST-6817-R5pdFtibJBFIJRqi60GI-cas",
        "idleTimeout": 1800,
        "sessionTimeout": 21600
    },
    "version": "1.0"
}
```

Note: If there is an error, check the Status of the request. Check the meaning of the status code. 200 means success. A 404 error may mean that the URL was entered incorrectly. A 401 or 403 error could indicate a problem with the authentication, so check that the credentials are entered correctly in the request body.

Part 2: Get a Service Ticket in Python

In this part, you will use Python to request a service ticket value from the APIC-EM.

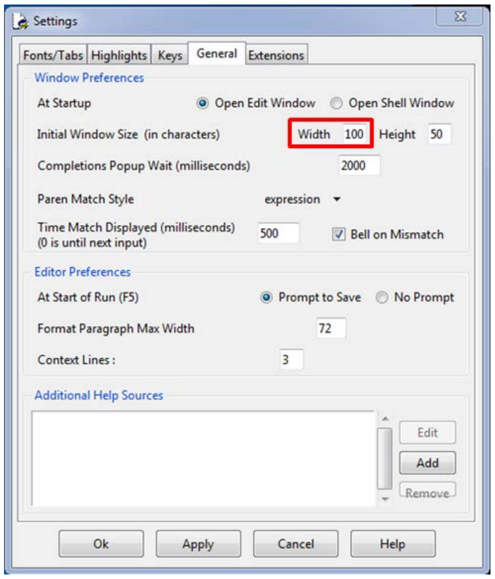
Step 1: Setup the Python script environment.

The IDLE shell window is a little narrow. In order to get better displays, configure IDLE as follows:

a. Open IDLE.

b. Click **Options > Configure IDLE > General** tab.

c. Change **Initial Window Size** to Width = **100** and click **Ok**.



Window Preferences

At Startup [✓ Open Edit Window] Open Shell Window

Initial Window Size (in characters) Width 100 Height 50

Completions Popup Wait (milliseconds) 2000

Paren Match Style expression

Time Match Displayed (milliseconds) (0 is until next input) 500 [✓ Bell on Mismatch]

Editor Preferences

At Start of Run (F5) [✓ Prompt to Save] No Prompt

Format Paragraph Max Width 72

Context Lines : 3

Additional Help Sources

Step 2: Import modules and disable SSL warnings.

a. Click **File > New File** to open IDLE Editor.

b. Save the file as **get_ticket.py**.

c. Enter the following commands to import the modules and disable SSL certificate warnings:

```
import json
import requests
requests.packages.urllib3.disable_warnings()
```

The **json** module includes methods that convert JSON data to Python objects and vice versa. The **requests** module has methods that will let us send REST requests to a URI.

Step 3: Build the request components.

Create a string variable to hold the API endpoint URI and two dictionaries, one for the request header and one for the body JSON. These are the same tasks you completed in the Postman application.

a. Create a variable named **api_url** and assign the appropriate URL to it, including the path to the API endpoint. Replace the URL with the URL of the sandbox you are assigned for this workshop.

```
api_url = "https://{YOUR-APICEM}.cisco.com/api/v1/ticket"
```

b. Create a dictionary variable named **headers** that has a single key for **content-type** and assign the key the value **application/json**.

```
headers = {
    "content-type": "application/json"
}
```

c. Create a dictionary variable named **body_json** that has two keys needed for authentication, **username** and **password**. Replace the username and password values with the credentials you are assigned for this workshop.

```
body_json = {
    "username": "!!!REPLACEME with the Username!!!",
    "password": "!!!REPLACEME with the Password!!!"
}
```

Step 4: Send the request.

You will now use the variables created in the previous step as parameters for the **requests.post()** method. This method actually sends the request over the Internet to the APIC-EM's API. You will assign the result of the request to a variable name **resp**. That variable will hold the JSON response from the API. If the request is successful, the JSON will contain the service ticket number.

a. Enter the following statement:

```
resp=requests.post(api_url,json.dumps(body_json),headers=headers,verify=False)
```

The various elements of this statement are:

Element	Explanation
`resp`	the variable to hold the response from the API.
`requests.post()`	the method that actually makes the request.
`api_url`	the variable that holds the URL address string
`json.dumps()`	a method from the requests module that converts Python dictionaries to JSON format.
`body_json`	the dictionary variable created to hold the authentication information
`headers=headers`	a parameter that is assigned to the headers variable
`verify=False`	disables verification of the SSL certificate when the request is made

b. Save your script and run it. There will not be any output yet but the script should run without errors. If not, review the steps and make locate and fix any errors.

Step 5: Evaluate the response.

Now the services ticket values can be extracted from the response JSON.

a. Display the status code for the request. The requests module has a built in object that contains the status code for the transaction.

```
print("Ticket request status: ", resp.status_code)
```

b. The response JSON is not compatible with Python dictionary and list objects so it is converted to Python format. Create a new variable called **response_json** and assign the variable **resp** to it adding the **json()** method to convert the JSON. The statement is as follows:

```
response_json = resp.json()
```

c. You can verify that your code returns the JSON in the IDLE Shell by temporarily adding a print statement to your script, as follows:

```
print(response_json)
```

d. Save and run your script. You should get output similar to the following although your service ticket number will be different:

```
{'response': {'serviceTicket': 'ST-5160-QHyLDw4TnICAzdNs63Vf-cas', 'idleTimeout':
1800, 'sessionTimeout': 21600}, 'version': '1.0'}
```

e. Create a variable named serviceTicket and assign the value of the serviceTicket key from the response key. Recall that you must first identify the response key and then you can access the serviceTicket key as follows:

```
serviceTicket = response_json["response"]["serviceTicket"]
```

f. Add a print statement to display the service ticket number, as follows:

```
print("The service ticket number is: ", serviceTicket)
```

g. Save and run your script. If you experience errors, check the code again.

Note: If you are stuck, compare your code to the **01_get_ticket_sol.py** file that is in the folder with the workshop files.

Step 6: Create a function from the program.

Rather than recreating code in every program that it is used in, you can import and refer to the same code repeatedly by calling a function that resides in a separate file. If the function code needs to be modified, it can be changed without directly impacting the code in other programs. Functions make your code modular and reusable. Putting your functions in a separate file that is imported as a module into a calling program puts the functions together in one place, which makes them easier to manage. In this step, you will create a Python function out of your **get_ticket.py** program so that it can called by other programs with a single statement.

A function requires the following:

- A line of code to define it,
- All code below the definition statement be indented at least four spaces (in addition to any other indentation),
- And an optional **return** statement that refers to the value that will be returned to the program that calls the function.

a. Open a new file in IDLE Editor and save it as **my_apic_em_functions.py**.

b. Copy all of the code from your **get_ticket.py** file and paste it into the new file.

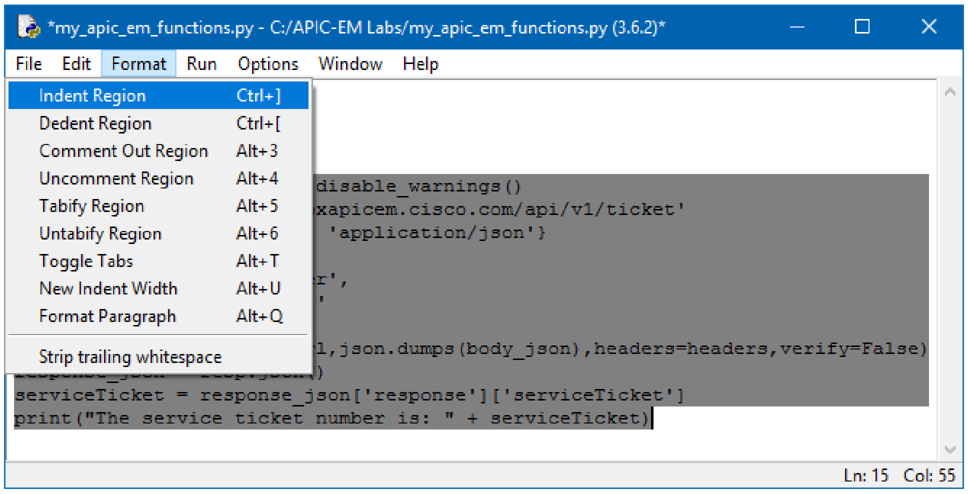
c. Enter a few blank lines below the **import** statements

d. Add the following statement below the import statements to define your code as the function **get_ticket()**. Be sure you include the colon.

```
def get_ticket():
```

e. Select all of the statements below the **def get_ticket():** line.

f. Click **Format > Indent Region**, as shown below, or press the TAB key to indent the region.



```
disable_warnings()
xapicem.cisco.com/api/v1/ticket'
 'application/json'}
r',
'
1,json.dumps(body_json),headers=headers,verify=False)
)
serviceTicket = response_json['response']['serviceTicket']
print("The service ticket number is: " + serviceTicket)
```

g. Finally, add a line at the end of the indented code. The service ticket number is returned to the program that calls the function. Add this line:

```
return serviceTicket
```

h. Save and run the **my_apic_em_functions.py** script. The file will run in the IDLE Shell, although nothing is displayed. Call your function by entering **get_ticket()** in the IDLE Shell. You should see output similar to the following:

```
>>> get_ticket()
The service ticket number is: ST-5274-Ey2TCHlMVqntwNqEkGzs-cas
'ST-5274-Ey2TCHlMVqntwNqEkGzs-cas'
>>>
```

i. If you have problems running your code at any time, refer to the appropriate solutions file for help if necessary.