Lab - Getting a Service Ticket with Python (Instructor Version)

**Instructor Note**: Red font color or Gray highlights indicate text that appears in the instructor copy only.

1. Objectives

Part 1: Get a service ticket in Postman

Part 2: Get a service ticket in Python

1. 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.

1. Required Resources

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

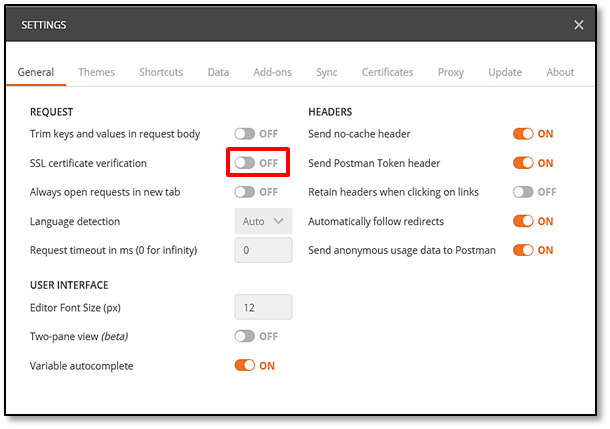
1. Get a Service Ticket in Postman

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

* + 1. Configure Postman

Configure a Postman setting.

* + - 1. Open Postman and create a new tab if necessary.
      2. Open the **File** menu and select **Settings**.
      3. Go to the **General** tab. Verify that **SSL certificate verification** is set to **OFF**.



* + 1. Select the method and Enter the Required URL
       1. Next to the URL box, select the request method as **POST**.
       2. Enter the URL for API endpoint: [**https://sandboxapicem.cisco.com/api/v1/ticket**](https://sandboxapicem.cisco.com/api/v1/ticket)**.**
    2. Enter the Header information
       1. Below the URL input field, select **Headers**.
       2. Under headers, click the first rectangle in the **Key** column.
       3. Enter **Content-Type**. Postman may help look this value up for you.
       4. In the value column enter **application/json**.
    3. Enter the Authentication Information

The **/ticket** endpoint requires that authentication information is carried in the body of the request. The username and password of **devnetuser** and **Cisco123!** Are used here, as they were at the sandbox login.

* + - 1. Select the **Body** tab that is next to the **Headers** tab.
      2. Click **Raw** to format the JSON data.
      3. Type the following exactly as it appears below.

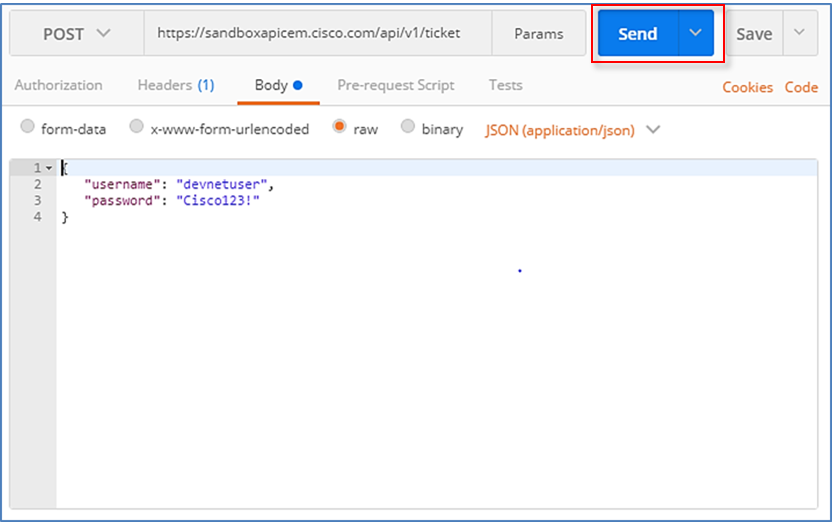
{

"username": "devnetuser",

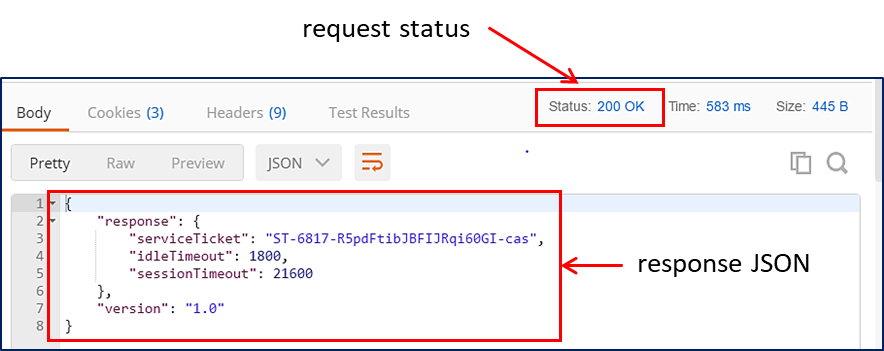
"password": "Cisco123"

}

Your screen should look like the one below.



* + 1. Send the Request
       1. You are now ready to send the request. Click the Send button that is near the URL entry field.
       2. After a brief delay, you should see the response JSON appear below the request information.



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.

The response consists of JSON data that includes the service ticket number and some other information about the service ticket. We will learn how to extract the service ticket number with Python and make it available to other requests that require it later in this lab.

Please keep Postman and this tab open. You will use it in the next lab.

1. Get a Service Ticket in Python

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

* + 1. Setup the Python Script Environment

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

* + - 1. Open IDLE.
      2. Go to the **Options** menu and choose **Configure IDLE**.
      3. Go to the **General** tab.
      4. Change **Initial Window Size** to Width = **100**.
      5. Click **Ok**.

In the new Python file, we import the modules we need, and disable SSL certificate warnings.

* + - 1. Type the following commands:

import json

import requests

request.packages.urllib3.disable\_warnings()

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

* + 1. Build the Request Components

We will create a string variable to hold the API endpoint URI and two dictionaries, one for the request header and one for the body JSON.

* + - 1. Assign the URI string to the post\_url variable.

post\_url = 'https://sandboxapicem.cisco.com/api/v1/ticket'

* + - 1. Create a dictionary that has a single key for content-type. Assign this key the value of **application/json**. Assign this dictionary to a variable called **headers**.

headers = {'content-type': 'application/json'}

* + - 1. Now create the dictionary that will be sent in the body for authentication. This dictionary has two keys, **username** and **password**. To make the dictionary more readable, we enter it in "Pretty" format. Assign the dictionary to the **body\_json** variable.

body\_json = {

'username': 'devnetuser',

'password': 'Cisco123!'

}

* + 1. Send the Request

You will now use the variables created in Step 2 as parameters for the **requests.post()** method. This method actually sends the request to over the Internet to the API. We assign the result of the request to a variable. That variable will hold the JSON response from the API. If the request is successful, the JSON will contain the service ticket number.

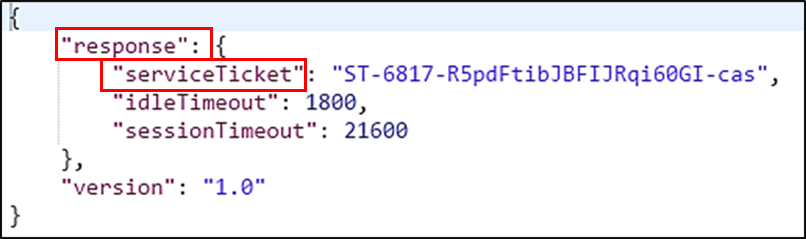
Enter the following statement

resp=requests.post(post\_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. |
| post\_url | the variable that holds the URI address string |
| json.dumps() | a method from the requests module that converts Python dictionaries to JSON format. |
| body\_json | the dictionary variable we created to hold the authentication information |
| headers=headers | a parameter that we assign the headers variable to |
| verify=False | disables verification of the SSL certificate when the request is made |

The JSON that is returned from the APIC-EM and held in the **resp** variable should resemble this:



* + 1. Evaluate the Response

We now need to extract the service ticket value from the response JSON.

* + - 1. The response JSON is not compatible with Python dictionary and list objects. We need to convert it to Python format. To do this, create a new variable called **response\_json** and assign the value of the converted JSON to it. The JSON in resp is converted using the **json()** method that is available to requests module response objects. The statement is as follows:

response\_json = resp.json()

* + - 1. Now we can access the values in the dictionary using Python. We need to refer to two keys in the statement that extracts the information we need. The first is the **response** key. The second is the **serviceTicket** key. We will assign this value to the variable **serviceTicket**. In essence we are creating a statement that says "Put the value of the **serviceTicket** key of the **response** key into the **serviceTicket** variable."

serviceTicket = response\_json['response']['serviceTicket']

* + - 1. Finally, we print out the value of the service ticket to indicate that the value has been successfully retrieved and extracted by using **print()** to display the variable. We add a descriptive string to make it clear to the user what the value is.

print("The service ticket number is: " + serviceTicket)

* + - 1. Save the Python file as **get\_ticket.py**. At the time the file is saved, Python debugs the code at a high level. If you receive an error on saving the file, check your code carefully, especially the punctuation and matching symbols, correct errors and try again.

Press **F5** to run the code. If you experience more errors, check the code again.

**Note:** If you are stuck, compare your code to the **get\_ticket\_sol.py** file that is in the folder with the workshop files.

* + 1. Create a Function from the Program

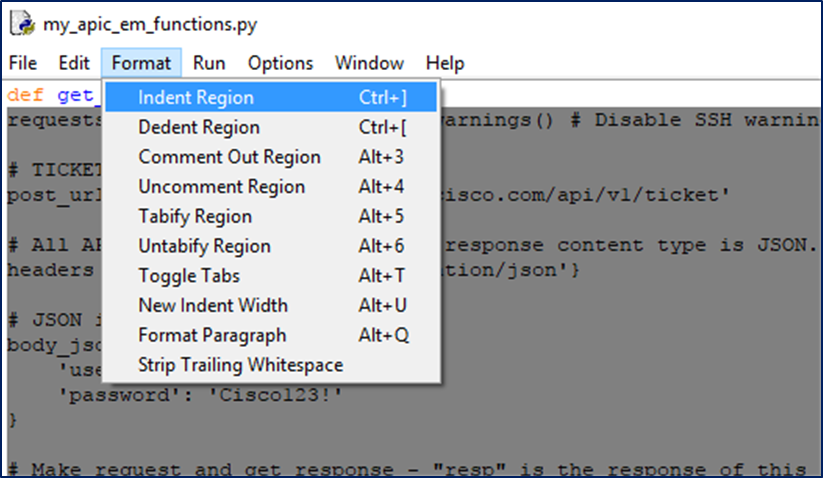
It is easy to create your own functions in Python. You can use Python functions to greatly simplify your code by replacing many lines of code with a single statement. Functions also allow you to make your code modular and reusable. This makes coding much more efficient. For example, rather than recreating code in every program that it is used in, you can import and refer to the same function repeatedly. In addition, if something needs to be changed, it can be changed in the function code often without an impact to the other programs that use the function code.

We will take the service ticket code that we have just created and create a function from it. The function will be stored in a separate Python file that we can import to any program that needs to use the function. A function requires a line of code to define it, that all code below the definition statement be indented at least four spaces, and an optional return statement that refers to the value that will be returned to the program that calls the function.

* + - 1. Create a new Python file, name it **my\_apic\_em\_functions.py,** and save it in the folder with your workshop files.
      2. Copy all of the code from your **get\_ticket.py** file and paste it into the new file.
      3. Enter a few blank lines after the **import** statements and enter the statement that indicates that the code below will be a function with the name **get\_ticket()**.

def get\_ticket()

* + - 1. Select all of the statements below the def get\_ticket() line.
      2. Go to the IDLE format menu and select **Indent Region**.



* + - 1. Finally, add a line at the end of the indented code. We want to return the service ticket number to the program that calls the function. Add this line:

return serviceTicket

* + - 1. Test your function by running your functions file in IDLE. The file will run, although nothing is displayed. At the shell prompt, type **get\_ticket()**. You should see a service ticket number appear, just as if the **get\_ticket.py** program were run.

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

You have completed your first network programmability with APIC-EM lab.