

Práctica calificada 2 - CC312

Sergio Sebastian Pezo Jimenez - 20224087G

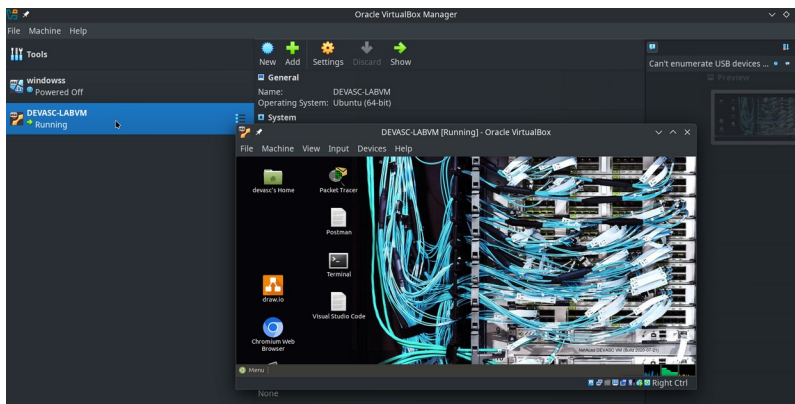
Índice

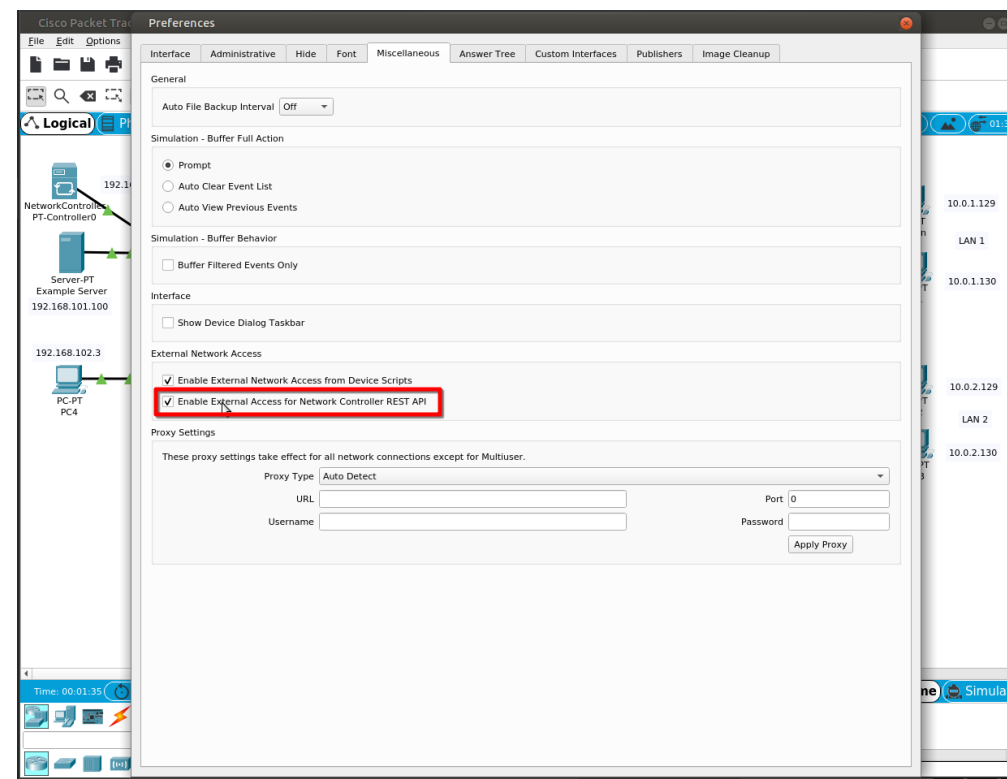
- Práctica calificada 2 - CC312
 - Índice
 - Desarrollo
 - Parte 1: Inicializamos la DEVASC VM.
 - Parte 2: Verificamos la conectividad externa con el Packet Tracer.
 - Parte 3: Solicitamos un Authentication Token con Postman
 - Parte 4: Enviamos solicitudes REST con PostMan
 - Parte 5: Enviamos solicitudes REST con VS Code
 - Parte 6: Enviamos solicitudes REST dentro del Packet Tracer
 - Conclusiones

Desarrollo

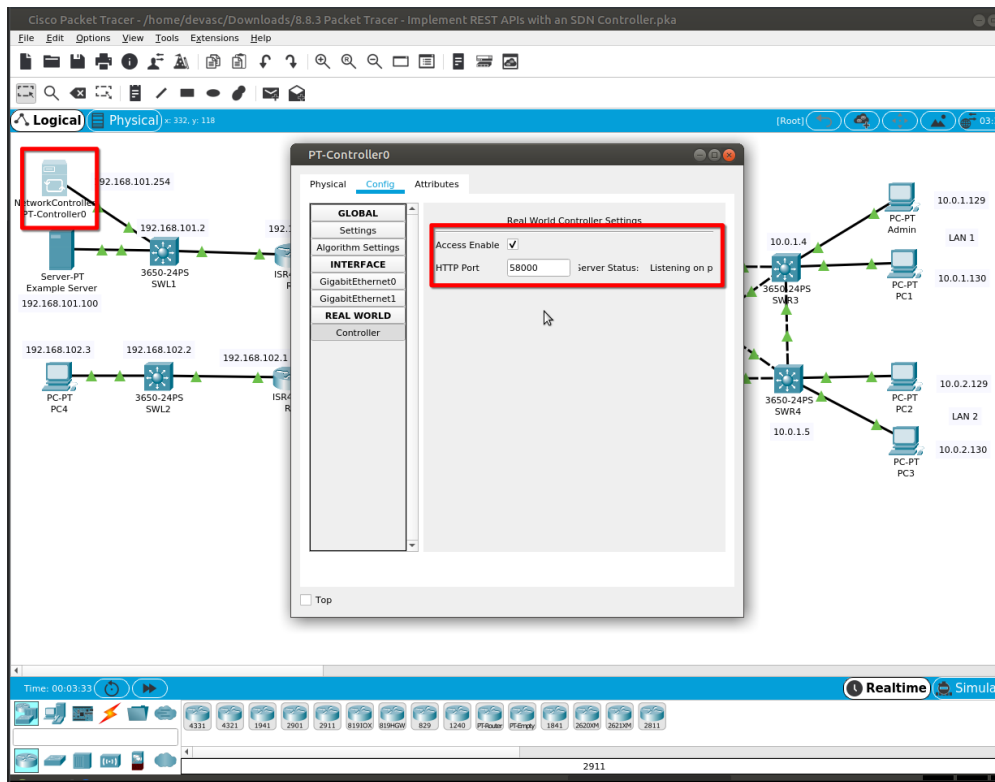
Parte 1: Inicializamos la DEVASC VM.

Inicialamos nuestra máquina virtual desde Virtual Box.

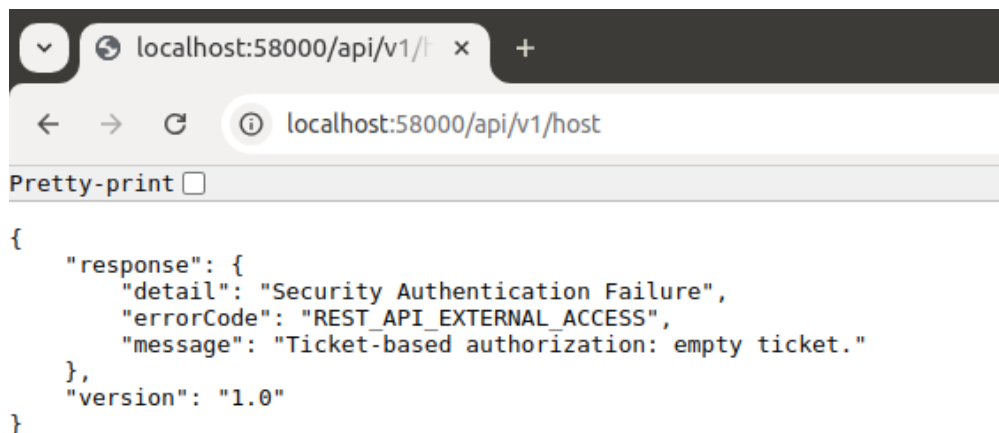




De igual para nuestro PT-Controller0, la cual accederemos mediante el puerto 58000.

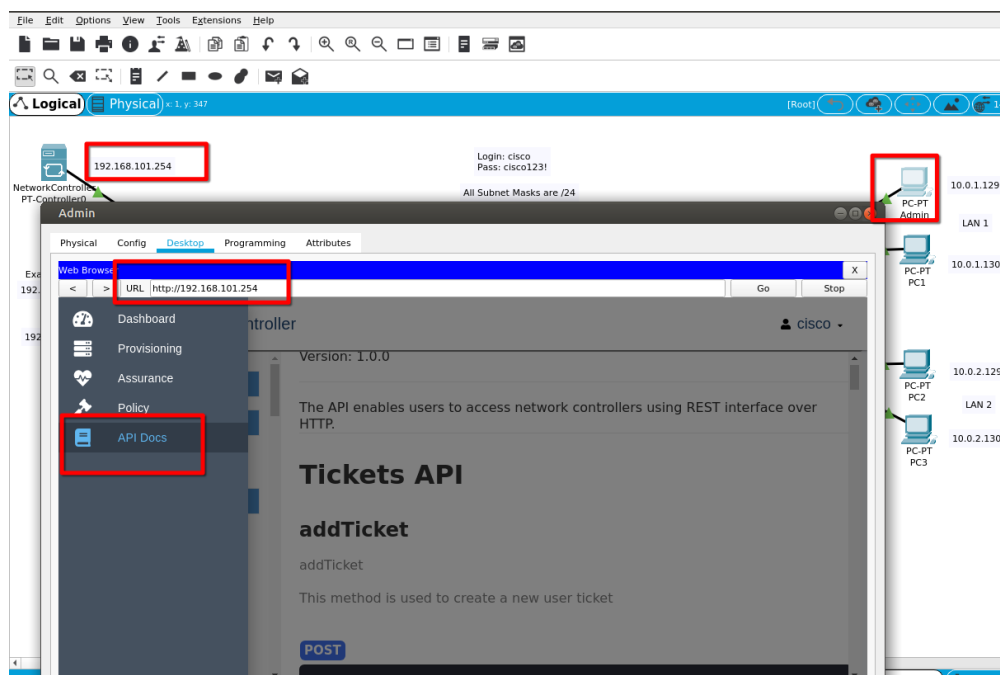


Verificamos si podemos acceder desde Chromium.

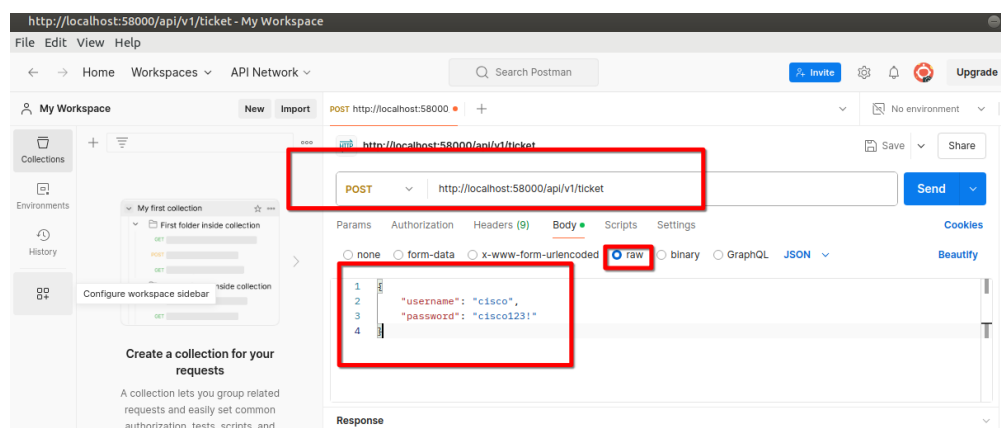


Parte 3: Solicitamos un Authentication Token con Postman

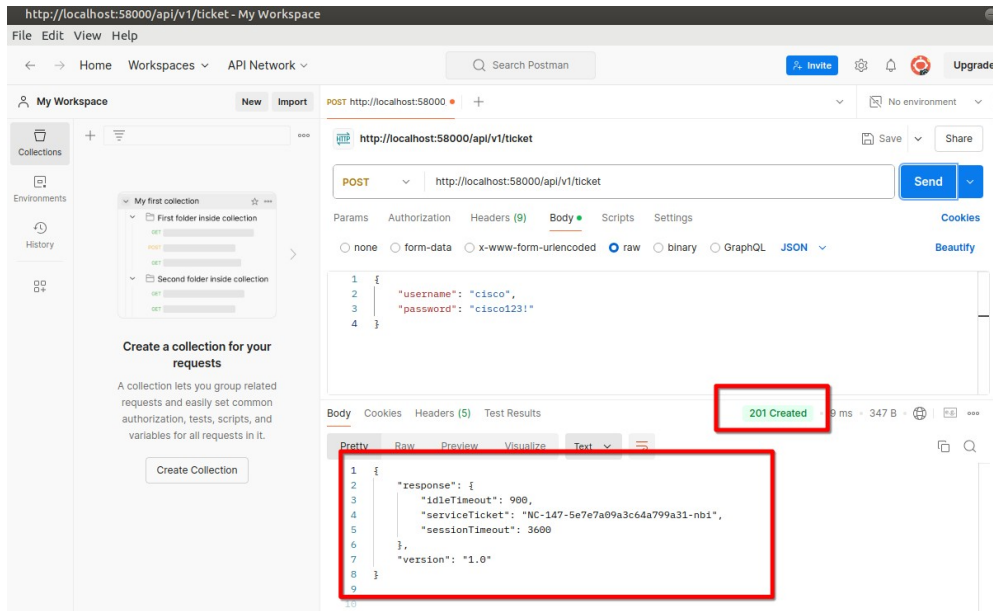
Para ver la documentación del PT-Controller0, abrimos el navegador desde la PC-Admin entrando a su ip 192.168.101.254, e iniciamos sesión, para irnos a la API docs.



A continuación creamos un petición POST a `http://localhost:58000/api/v1/ticket` en postman

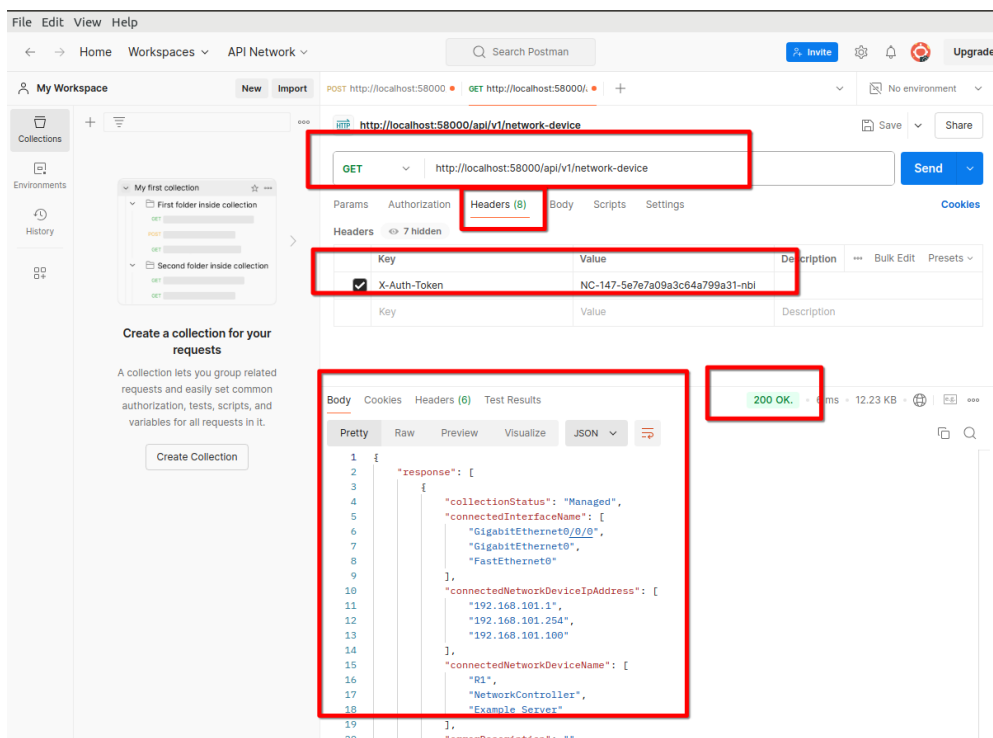


Al hacer la petición recibimos una respuesta `201` así como nuestro ticket:

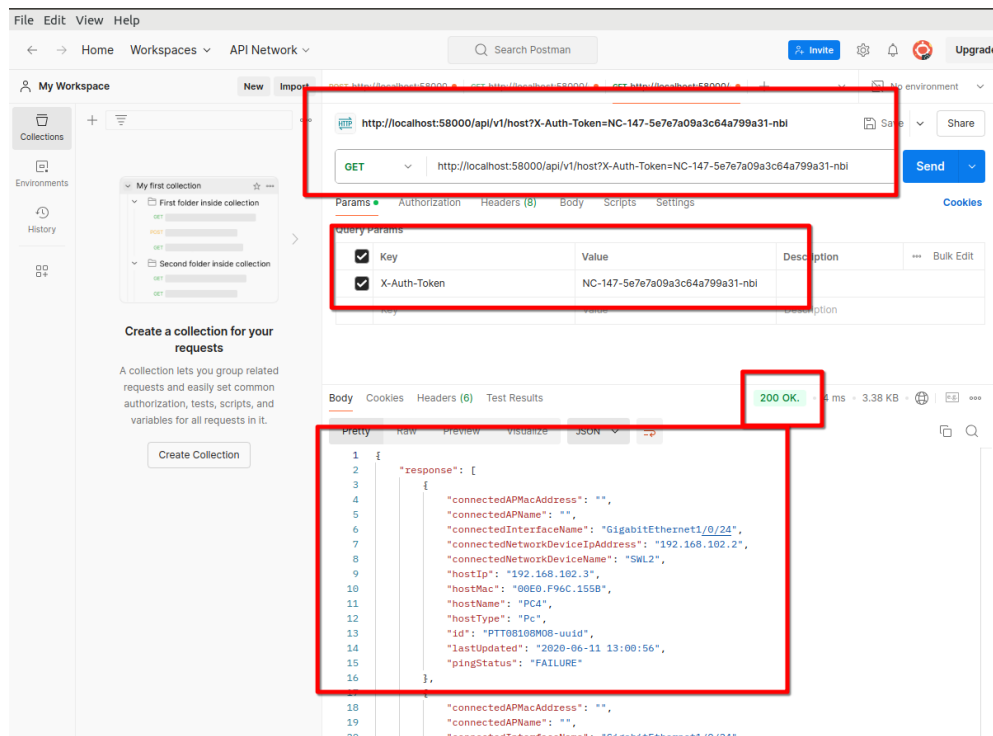


Parte 4: Enviamos solicitudes REST con PostMan

Creamos una solicitud GET para todos los dispositivos en la red, con nuestro ticket previamente creado como header `X-Auth-Token`. Para obtener como respuesta un estado `200`.



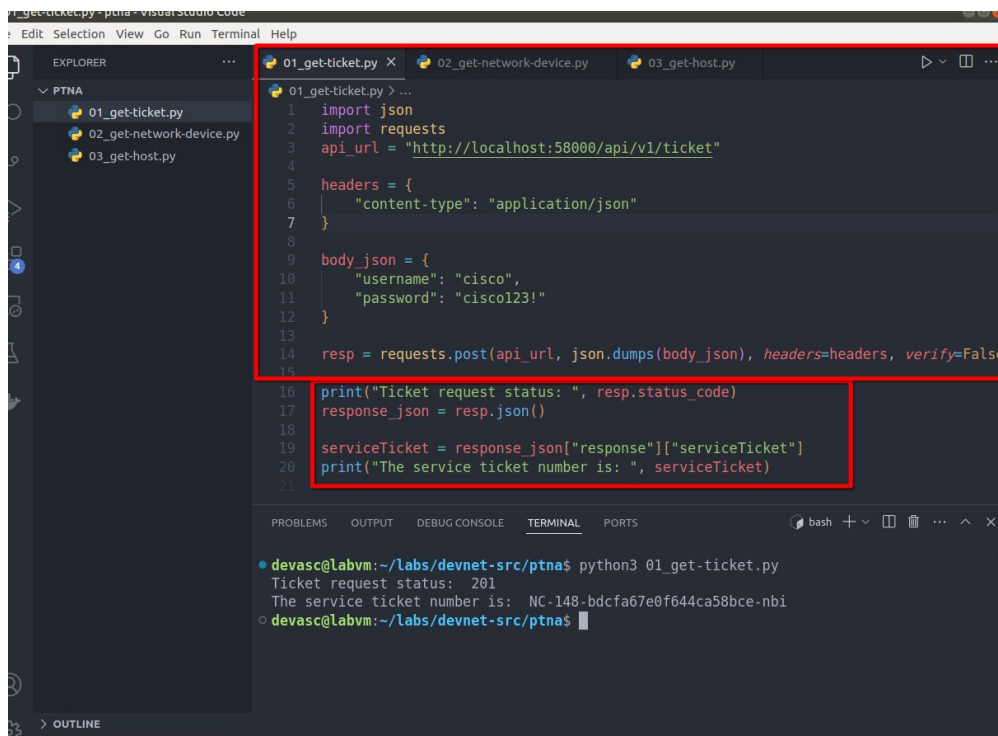
Duplicamos la petición `GET` , pero ahora a `http://localhost:58000/api/v1/host` para el PT-Controller0, obteniendo una respuesta `200` .



Ahora si podemos cerrar Postman.

Parte 5: Enviamos solicitudes REST con VS Code

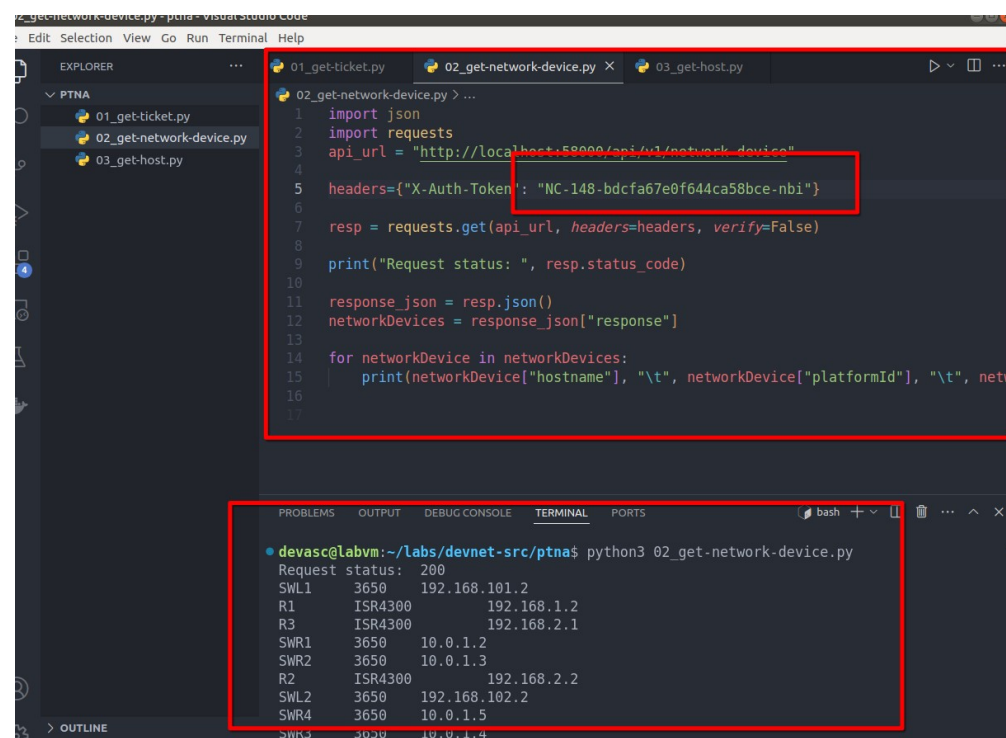
Abrimos VS Code en el directorio `devnet-src/ptna`. Y vamos a ejecutar `01_get-ticket.py` script, donde solicitaremos un nuevo ticket pero ahora en python, y ya directamente imprime el ticket.



```
01_get-ticket.py > ...
1 import json
2 import requests
3 api_url = "http://localhost:58000/api/v1/ticket"
4
5 headers = {
6     "content-type": "application/json"
7 }
8
9 body_json = {
10     "username": "cisco",
11     "password": "cisco123!"
12 }
13
14 resp = requests.post(api_url, json.dumps(body_json), headers=headers, verify=False)
15
16 print("Ticket request status: ", resp.status_code)
17 response_json = resp.json()
18
19 serviceTicket = response_json["response"]["serviceTicket"]
20 print("The service ticket number is: ", serviceTicket)
21
```

```
devasc@labvm:~/labs/devnet-src/ptna$ python3 01_get-ticket.py
Ticket request status: 201
The service ticket number is: NC-148-bdcfa67e0f644ca58bce-nbi
devasc@labvm:~/labs/devnet-src/ptna$
```

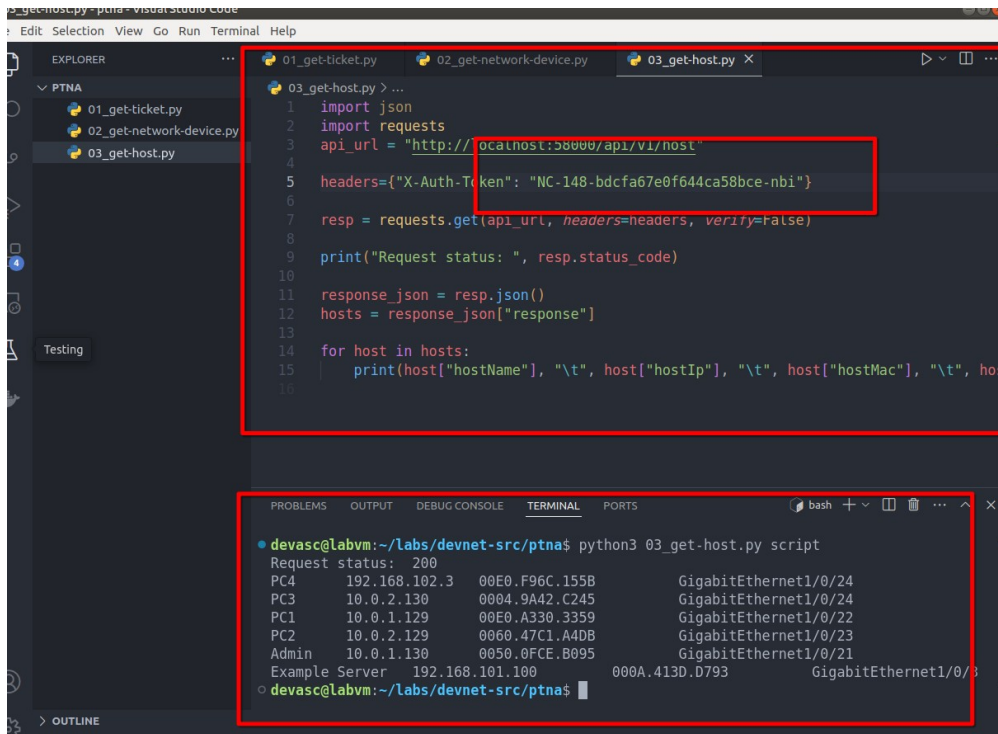
Continuando, ejecutaremos un script en `02_get-network-device.py` para listar todos los dispositivos de red, pero primero tenemos que ingresar nuestro ticket generado en el paso anterior.



```
02_get-network-device.py > ...
1 import json
2 import requests
3 api_url = "http://localhost:58000/api/v1/network_device"
4
5 headers={"X-Auth-Token": "NC-148-bdcfa67e0f644ca58bce-nbi"}
6
7 resp = requests.get(api_url, headers=headers, verify=False)
8
9 print("Request status: ", resp.status_code)
10
11 response_json = resp.json()
12 networkDevices = response_json["response"]
13
14 for networkDevice in networkDevices:
15     print(networkDevice["hostname"], "\t", networkDevice["platformId"], "\t", networkDevice["ipAddress"])
16
17
```

```
devasc@labvm:~/labs/devnet-src/ptna$ python3 02_get-network-device.py
Request status: 200
SWL1 3650 192.168.101.2
R1 ISR4300 192.168.1.2
R3 ISR4300 192.168.2.1
SWR1 3650 10.0.1.2
SWR2 3650 10.0.1.3
R2 ISR4300 192.168.2.2
SWL2 3650 192.168.102.2
SWR4 3650 10.0.1.5
SWR3 3650 10.0.1.4
```

Ahora ejecutaremos `03_get-host.py` script, que es un script para list los host de la red.



The image shows a Visual Studio Code editor window with three tabs: `01_get-ticket.py`, `02_get-network-device.py`, and `03_get-host.py`. The `03_get-host.py` script is open in the editor. It imports `json` and `requests`, sets an `api_url` to `http://localhost:5000/api/v1/host`, and defines headers for an `X-Auth-Token`. The script then makes a GET request, prints the status code, and iterates over the response to print host details.

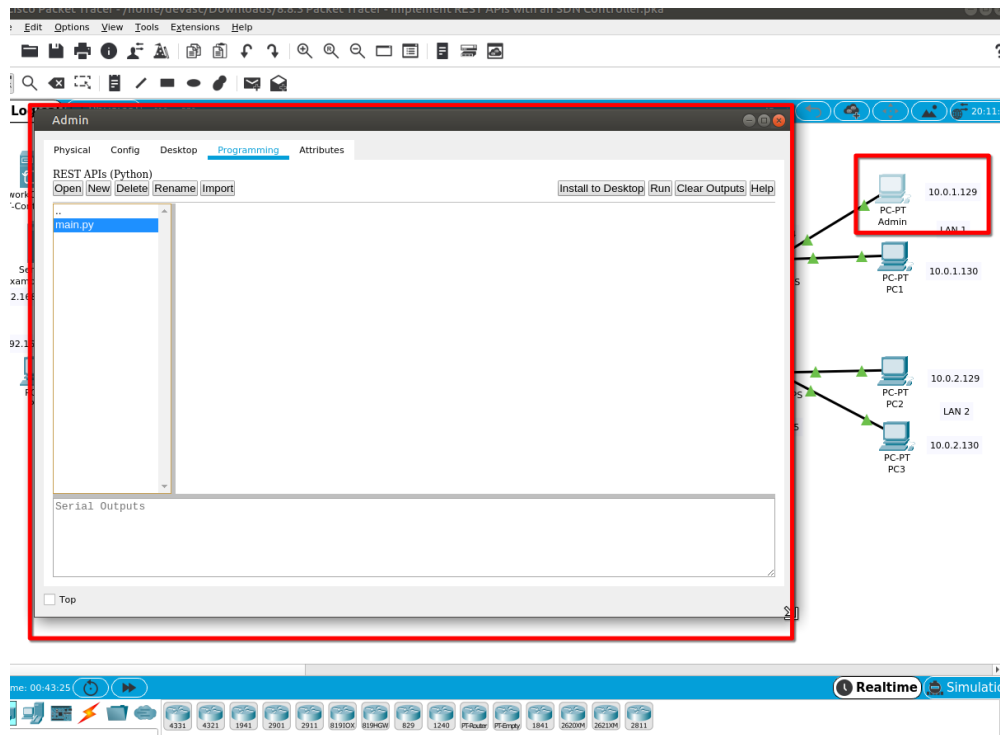
```
1 import json
2 import requests
3 api_url = "http://localhost:5000/api/v1/host"
4
5 headers={"X-Auth-Token": "NC-148-bdcfa67e0f644ca58bce-nbi"}
6
7 resp = requests.get(api_url, headers=headers, verify=False)
8
9 print("Request status: ", resp.status_code)
10
11 response_json = resp.json()
12 hosts = response_json["response"]
13
14 for host in hosts:
15     print(host["hostName"], "\t", host["hostIp"], "\t", host["hostMac"], "\t", host["hostInterface"])
16
```

Below the editor, the terminal shows the command `python3 03_get-host.py script` being executed. The output displays the request status (200) and a list of hosts with their names, IP addresses, MAC addresses, and interfaces.

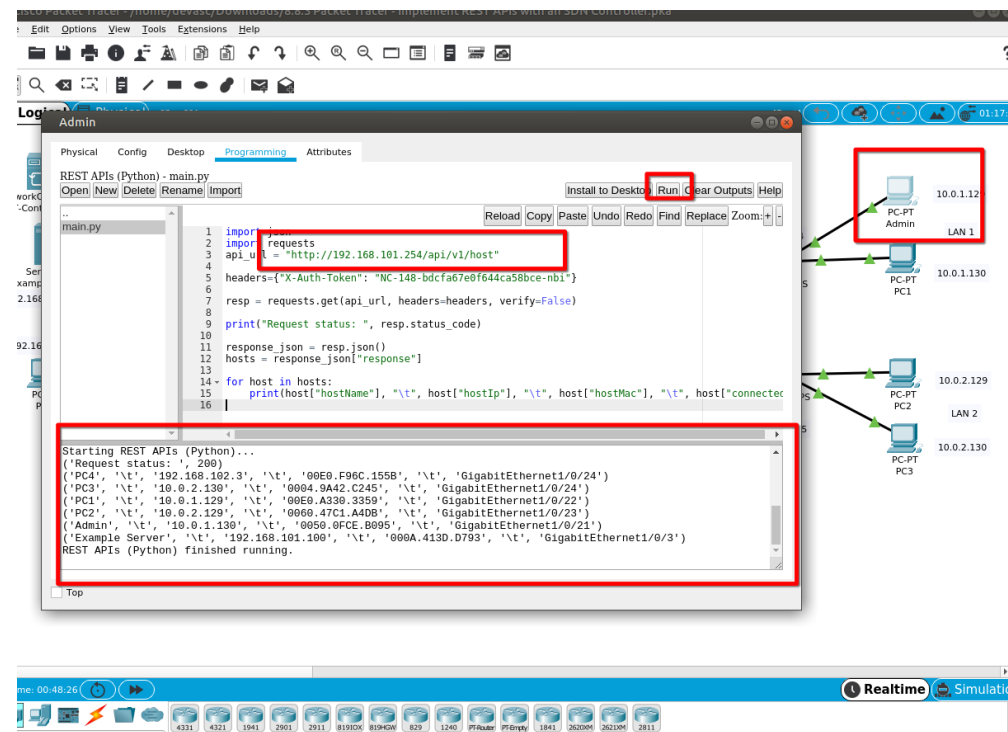
```
devasc@labvm:~/labs/devnet-src/ptna$ python3 03_get-host.py script
Request status: 200
PC4      192.168.102.3    00E0.F96C.155B    GigabitEthernet1/0/24
PC3      10.0.2.130       0004.9A42.C245    GigabitEthernet1/0/24
PC1      10.0.1.120       00E0.A330.3359    GigabitEthernet1/0/22
PC2      10.0.2.129       0060.47C1.A4DB    GigabitEthernet1/0/23
Admin    10.0.1.130       0050.0FCE.B095    GigabitEthernet1/0/21
Example Server 192.168.101.100 000A.413D.D793    GigabitEthernet1/0/3
```


Parte 6: Envamos solicitudes REST dentro del Packet Tracer

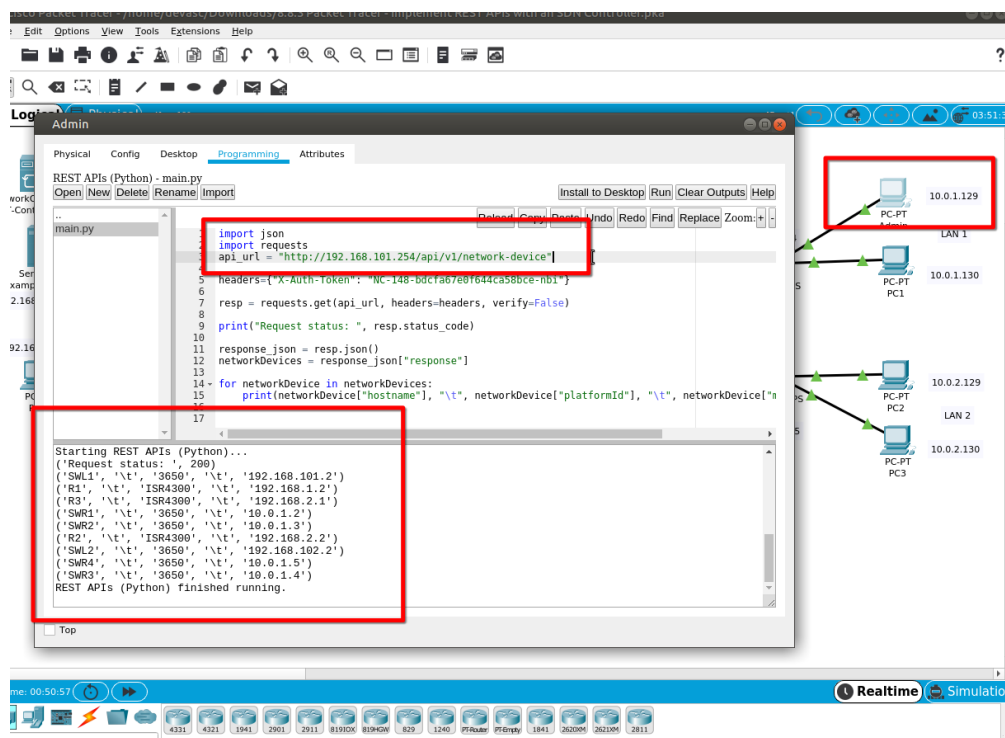
Primero creamos un proyecto en la PC-Admin de python



Copiamos el código de 03_get-host.py , cambiando la URL, obteniendo un respuesta 200 .



De igual manera para el script en `02_get-network-device.py`.



Finalizando así nuestro laboratorio.

Conclusiones

En esta práctica calificada se logró:

- Verificar la conectividad externa, asegurando que todos los componentes de red están correctamente configurados y accesibles.
- Solicitar un Authentication Token con Postman, permitiendo la interacción con la API del PT-Controller0.
- Enviar solicitudes REST con Postman, entendiendo cómo interactuar con APIs y verificar la conectividad y respuesta de los dispositivos de red.
- Enviar solicitudes REST con VS Code, automatizando las interacciones con la API usando scripts en Python.
- Enviar solicitudes REST dentro del Packet Tracer, integrando scripts de Python directamente en un entorno de simulación de red.

Este laboratorio nos deja un valor significativo al enseñarnos cómo configurar y verificar entornos de red, interactuar con APIs mediante herramientas como Postman y VS Code, y automatizar tareas de red usando Python.