Automatización de la CLI

Para automatización simple usando una línea de comando remota como lo es Telnet o SSH, Administradores de red han usado distintos métodos y técnicas desde hace mucho tiempo. Inicialmente scripts automatizan con valores de string que se trasladan a la CLI. Con la evolución de Python. Netmiko un módulo para python ah emergido como una herramienta open source mantenida en Github.com, Esta herramienta provee una interfaz similar a los scripts basados en respuestas de String.

Pasos para automatizar una CLI.
1)se importa la librería o módulo netmiko y de esta el método ConnectHandler.
2)se crea una variable que usará el método Connect Handler.
3)dentro del método ConnectHandler ingresamos los parámetros necesitados.
4)luego creamos otra variable (output) y invocamos la variable sshCli + la función send_command dentro de la función send_command ingresamos el comando “show Ip int brief”.
5)usamos la funcion print() desplegamos en pantalla el contenido de output.

```
C:\Users\q669YSCG>python -i
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 22:39:24) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> from netmiko import ConnectHandler
>>> sshCli = ConnectHandler(
... device_type="cisco_ios",
... host="10.10.20.48",
... port=22,
... username="developer",
... password="C1sco12345"
... )
>>> output = sshCli.send_command("show ip interface brief")
>>> print(output)
Interface              IP-Address      OK? Method Status                Protocol
GigabitEthernet1       10.10.20.48     YES NVRAM  up                    up
GigabitEthernet2       unassigned      YES NVRAM  administratively down down
GigabitEthernet3       unassigned      YES NVRAM  administratively down down
>>>
```

Aclaración: sshCli es una variable que está usando el método ConnectHandler de la librería netmiko. Output también es una variable usando otro método llamado send_comand de la librería netmiko.

Configurando una interfaz a través de un Script.

Ahora veremos cómo configurar una interfaz a través de un script. Los pasos son los siguientes:

1) Crear una lista llamada config_command
2) Ingresar los valores de la lista en cadena de caracteres
3) Usar la variable output antes creada y re definirla
4) Dentro de output llamar la variable sshCli con el método send_config_set y dentro de los paréntesis del método ingresar la variable que tiene los parámetros a configurar -> config_command

```
>>> config_command = [
... "int loopback 1",
... "ip address 7.7.7.7 255.255.255.0",
... "description Created with cli",
... ]
>>> output = sshCli.send_config_set(config_command)
```

Desplegar información de un script de configuración.

Como podemos ver en la imagen usamos la función print() y desplegamos el contenido de output.

```
>>> print(output)
config term
Enter configuration commands, one per line.  End with CNTL/Z.
csr1000v-1(config)#int loopback 1
csr1000v-1(config-if)#ip address 7.7.7.7 255.255.255.0
csr1000v-1(config-if)#description Created with cli
csr1000v-1(config-if)#end
csr1000v-1#
>>>
```

Si deseamos ver la interfaz creada hacemos uso de output y utilizamos dentro de output el método send_command + entre paréntesis “show ip interface brief”

```
>>> output = sshCli.send_command("show ip interface brief")
>>> print(output)
Interface              IP-Address      OK? Method Status                Protocol
GigabitEthernet1       10.10.20.48     YES NVRAM  up                    up
GigabitEthernet2       unassigned      YES NVRAM  administratively down down
GigabitEthernet3       unassigned      YES NVRAM  administratively down down
Loopback1              7.7.7.7         YES manual up                    up
>>>
```

API's

Una API permite que una parte de software se pueda comunicar con otra. Podemos comparar la analogía de un toma corriente a una API. Que haces para conectar tu Laptop al toma corriente? (si no existiera el enchufe que conocemos hoy en dia?)

-Abrir la pared
-Pelar los Cables
-Dividir los cables
-Entender todos los cables en la pared.

Una API define como un programador escribe una pieza de código para poder comunicarse con aplicaciones existentes o inclusive construir nuevas aplicaciones.

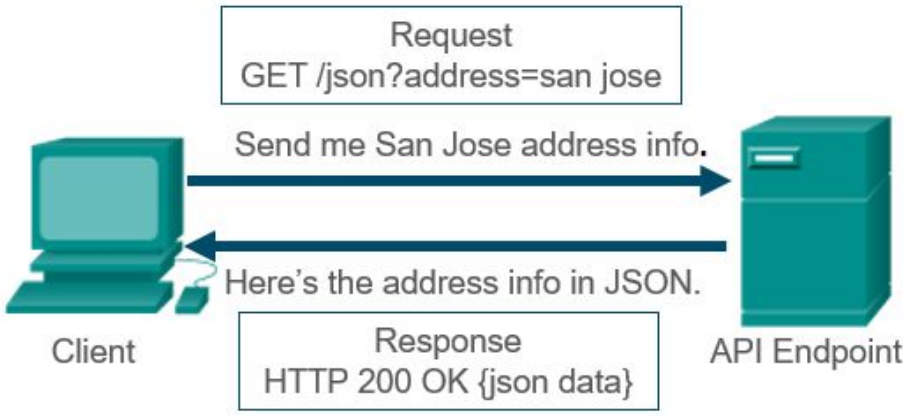
Servicios Web usando HTTP

Figure 1: HTTP Process for Requesting a Web Page

Web browsers usan HYPERTEXT TRANSFER PROTOCOL (HTTP) para poder obtener(GET) una pagina web. Si la solicitud es exitosa (HTTP status code 200), El servidor Web responde al GET request con un HYPERTEXT MARKUP LANGUAGE(HTML) codificado en la pagina web como se ve en la imagen de arriba.

RESTful API usando HTTP

Representation state Transfer(REST) APIs usan HTTP para tener una interfaz con servicios RESTful. El request pide por un formato json (JavaScript Object Notation(JSON))

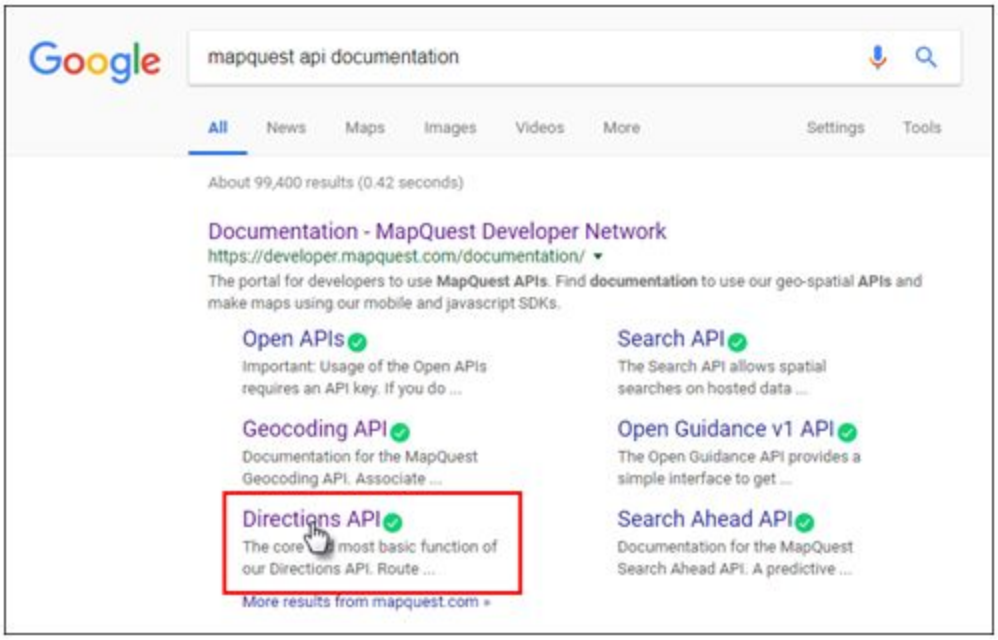


Anatomia de un RESTful request.

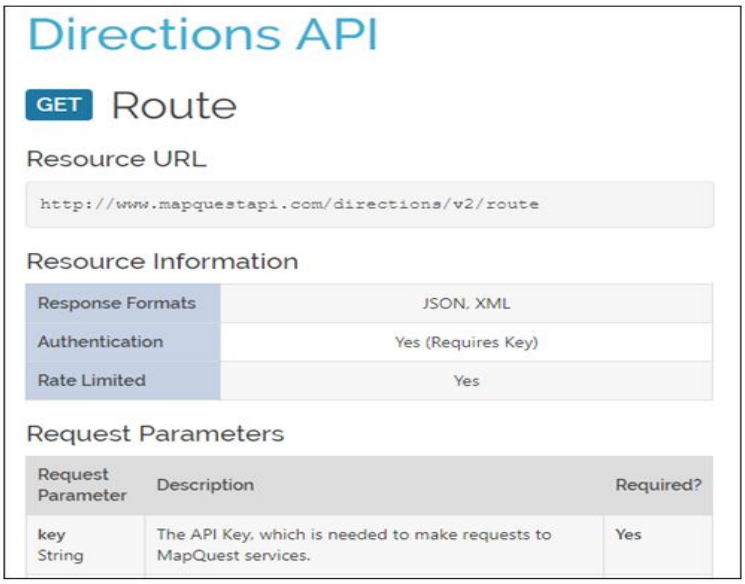
- API SERVER: la URL de el servidor que responde al REST request.
- RESOURCES: Especifica la API que esta siendo pedida(request)
- FORMAT: usualmente en JSON o XML
- PARAMETERS: especifica que data se esta pidiendo(request)

Documentacion API

Para poder tener éxito construyendo un request API, Se debe seguir la documentación API Puedes encontrar una variedad de esta información usando internet.



Usualmente la documentación API siempre específica:
- El formato pedido(JSON, XML, o TEXTO)
- Los parametros del request
- El formato de respuesta.



Autenticar una RESTful request

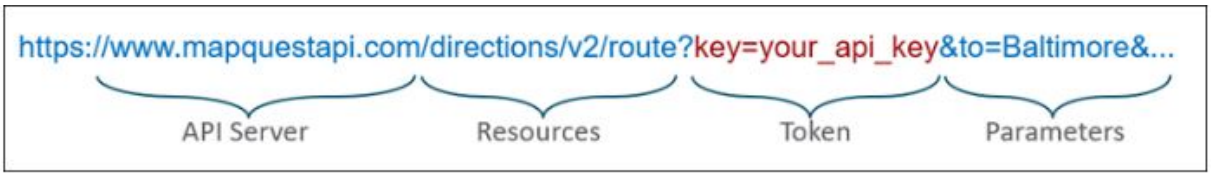
Para construir una aplicación necesitamos obtener un API KEY de una pagina developer del producto deseado. La autenticación se lleva de 4 formas.

-NONE: API es pública y todos pueden hacer un request

-BASIC HTTP: se necesita username y password

-TOKEN: una SECRET KEY se obtiene del Web API developer portal.

-OPEN AUTHORIZATION(**OAuth**): un estándar para obtener un Access token de un proveedor. El token se lleva en cada llamada API.



Creando nuestro primer API con MAPQUEST

En esta sección crearemos una aplicación que muestre JSON data, Esto desde direcciones de MAPQUEST API, Analizaremos la data y la manipularemos para tener un output para el end user.

Documentacion API: https://developer.mapquest.com/documentation/directions-api/route/get/

Para poder construir esta aplicación, Debemos completar lo siguiente.

- Obtener una MAPQUEST API key
- Importar modulos necesarios
- Crear variables API request y construir una URL.
- Darle user input como funcionalidad
- Agregar una QUIT/EXIT instrucción.

Primer paso: debemos ir al sitio developer crear una cuenta y obtener una llave/key, de la cual debemos usar la consumer Key.

Segundo paso: Importamos librerías urllib.parse y requests, urllib nos brinda funciones para analizar y manipular la data JSON. y requests nos brinda funciones para obtener JSO data de una URL.

```
import urllib.parse
import requests
```

Tercer Paso: debemos crear las variables que nos ayudarán a construir una URL

```
main_api = "https://www.mapquestapi.com/directions/v2/route?"
orig = "washington"
dest = "Baltimaore"
key = "yPM6tDNg7nm5QGKZbx8HYjNNU1GnktEM"
```

Main_api: es la URL principal que accedemos.
Orig: el parámetro que indica el punto de origen.
Dest: el parámetro que indica el punto destinación.
Key: es el MAPQUEST API key que obtuvimos del developer site.

Cuarto Paso: Creamos la variable URL

```
url = main_api + urllib.parse.urlencode({"key": key, "from":orig, "to":dest})
```

Aca podemos ver como dentro de URL estamos concatenando valores anteriormente declarados en el script.

Quinto paso: ahora ya estamos listos para poder crear el Request. Para eso creamos la variable json_data esta variable va hacer uso del método GET del módulo requests y va especificar JSON como el formato pedido.

```
json_data = requests.get(url).json()
```

Para saber si el GET request funcionó podemos hacer uso de la función print() y ver el output.

```
print(json_data)
```

Ahora que vemos que el JSON está funcionando, podemos empezar a agregar más funcionalidad a la aplicación.

Sexto Paso: eliminamos print(json_data) ya que no es necesario más. Ahora podemos usar la función print() para mostrar la información obtenida por el método GET en la variable json_data.

```
print("URL: " + (url))
```

También creamos la variable json_status que tendrá los valores `["info"]["statuscode"]`
De la variable json_data

Septimo paso: añadimos una condicional que analiza el status code del API request.

```
print("URL: " + (url))

json_data = requests.get(url).json()
json_status = json_data["info"]["statuscode"]

if json_status == 0:
    print("API Status: " + str(json_status) + " = A successful route call.\n")
```

Challenge: Basado en lo que aprendimos hasta ahora mejora la aplicación y añade opciones para tener user input y también agrega la opción quit con un condicional para terminar la aplicación.

```
#Replace "your_api_key" with your MapQuest API key

import urllib.parse
import requests

main_api = "https://www.mapquestapi.com/directions/v2/route?"
key = "your_api_key"

while True:
    orig = input("Starting Location: ")
    dest = input("Destination: ")
    url = main_api + urllib.parse.urlencode({"key": key, "from":orig, "to":dest})
    print("URL: " + (url))

    json_data = requests.get(url).json()
    json_status = json_data["info"]["statuscode"]

    if json_status == 0:
        print("API Status: " + str(json_status) + " = A successful route call.\n")
```

Challenge Resuelto

```
#Replace "your_api_key" with your MapQuest API key

import urllib.parse
import requests

main_api = "https://www.mapquestapi.com/directions/v2/route?"
key = "your_api_key"

while True:
    orig = input("Starting Location: ")
    if orig == "quit" or orig == "q":
        break
    dest = input("Destination: ")
    if dest == "quit" or dest == "q":
        break

    url = main_api + urllib.parse.urlencode({"key": key, "from":orig, "to":dest})
    print("URL: " + (url))

    json_data = requests.get(url).json()
    json_status = json_data["info"]["statuscode"]

    if json_status == 0:
        print("API Status: " + str(json_status) + " = A successful route call.\n")
print("done done!")
```