Автоматизация 4. Netmiko

Предыдущие статьи цикла

- Автоматизация работы сетевого администратора.
- Автоматизация работы сетевого администратора, часть 2. Практические кейсы с Telnetlib.
- Автоматизация работы сетевого администратора. Часть 3. Знакомство с Paramiko

Приветствую читателей цикла об автоматизации работы с сетевым оборудованием, после чуть затянувшегося перерыва мы продолжаем писать о различных программных инструментах . В конце предыдущей статьи, в которой мы познакомились с Paramiko, был упомянут модуль, разработанный сетевыми инженерами для сетевых инженеров.

Введение

Итак, Netmiko - это Python библиотека для работы с сетевыми устройствами, в качестве инициализации соединений она использует Paramiko, но содержит много встроенных методов и классов "под капотом".

Библиотека не входит в список стандартных модулей для Python 3, поэтому ее требуется установить:

```
pip install netmiko
```

В первую очередь рекомендовано ознакомиться со списком поддерживаемых сетевых ОС, из стабильных это:

- Arista vEOS
- Cisco ASA
- Cisco IOS
- Cisco IOS-XE
- Cisco IOS-XR
- Cisco NX-OS
- Cisco SG300
- HP ProCurve
- Juniper Junos
- Linux

Кратко об истории создания:

Разработчики, в частности сетевые инженеры, часто сталкивались с архитектурными (системными) ограничениями того или иного сетевого оборудования при написание различных скриптов, что не позволяло унифицировать его применение и вызывало ошибки выполнения.

Например, HP ProCurve свитчи выводят символы в ANSI-кодировке или Cisco требует дополнительные режимы в терминале (enable, configure terminal) и пр. Это побудило написать библиотеку Netmiko, где уже будет задана логика взаимодействия с сетевым устройством в зависимости от его ОС.

Синтаксис и логика

Напомню, в Python3 есть удобная возможность тестировать команды, для этого достаточно лишь вызвать сам интерпретатор:

```
python3
```

Первично, нужно импортировать функцию ConnectHandler и подготовить переменную типа словарь, в котором будут заданы необходимые параметры для SSH-подключения.

```
In [1]: from netmiko import ConnectHandler

In [2]: cisco = {
   ...:   "device_type": "cisco_ios",
   ...:   "host": "cisco.domain.com",
   ...:   "username": "admin",
   ...:   "password": "cisco123",
   ...:   }
```

После того, как определили "device_type": "cisco_ios", можем создавать соединение с устройством. Список поддерживаемых видов ОС доступен по ссылке.

Для удобства передадим функцию в переменную "net_connect", ее будет использовать далее при отправке команд и т.д.

In [3]: net_connect = ConnectHandler(cisco)**

С помощью ** мы передаем ключ-значение в функцию ConnectHandler, более подробно о теории по[ссылке](). Альтернативно возможно отправить параметры функции вручную:

net_connect2 = ConnectHandler(device_type="cisco_ios", host="cisco.domain.com", username="admin", password="cisco123")

После создания переменной "net_connect" и загрузки в нее параметров, можно убедиться что SSH-соединение установлено:

In [5]: net_connect.find_prompt()
Out[5]: "cisco3#"

Получен ответ (Out[5]:) от сетевого устройства, сессия активна, находимся в режимеenable (Cisco OS).

Сбор данных с одного устройства

В качестве первой задачи после подключения, рассмотрим получение вывода результата выполнения команды на экране пользователя (инициатора скрипта). Благодаря прописанной логике, в Netmiko достаточно вызвать следующий метод:

In [6]: output = net_connect.send_command("show ip int brief")
In [7]: print(output)

```
Interface              IP-Address      OK? Method Status                Protocol
GigabitEthernet0/0/0   10.10.10.22     YES manual up                    up
GigabitEthernet0/0/1   unassigned      YES NVRAM  administratively down down
GigabitEthernet0/1/0   unassigned      YES unset  down                  down
GigabitEthernet0/1/1   unassigned      YES unset  down                  down
GigabitEthernet0/1/2   unassigned      YES unset  down                  down
GigabitEthernet0/1/3   unassigned      YES unset  down                  down
Vlan1                  unassigned      YES unset  up                    down
```

Что произошло внутри ? В строке 6 была определена переменная"output", которой был передан класс"net_connect" - с данными о подключение и вызван метод "send_command".

Со стороны сетевого устройства: произошла успешная инициализация SSH-соединения, вход в режим enable и отправка команды "show ip int brief" в терминал.Далее осталось лишь записать вывод в переменную "output"и вывести ее на экран.

Отработает ли метод, если внутри команды задать еще одно условие для выполнения?

In [8]: output = net_connect.send_command("show run | inc logging")
In [9]: print(output)

```
 logging synchronous
```

В In [8] передана команда с просмотром текущего конфига (show run) и ее вывод передан в конвейер Pipeline для команды "inc logging". Соответственно, результат представлен на скриншоте, запрашиваемый параметр выведен.

Сбор данных с нескольких устройств

После того, как рассмотрели сбор данных с одного устройства, выполним задачу по опросу нескольких сетевых устройств, причем в топологии будут различные вендоры (сетевые ОС).

```
In [1]: from netmiko import ConnectHandler #Импорт библиотеки
"Создание словарей с информацией об устройствах"

In [2]: cisco3 = {
   ...:     "device_type": "cisco_ios",
   ...:     "host":   "cisco3.domain.com",
   ...:     "username": "admin",
   ...:     "password": "cisco123",
   ...: }
   ...:
In [3]: cisco_asa = {
   ...:     "device_type": "cisco_asa",
```

```
   ...:     "host": "10.10.10.88",
   ...:     "username": "admin",
   ...:     "password": "cisco123",
   ...:     "secret": "cisco123",
   ...: }
   ...:
In [4]: cisco_xrv = {
   ...:     "device_type": "cisco_xr",
   ...:     "host": "10.10.10.77",
   ...:     "username": "admin",
   ...:     "password": "cisco123",
   ...: }
```

Далее нам понадобится уже известный цикл FOR, с помощью которого мы сможем последовательно подключаться к каждому из сетевых устройств и предоставлять вывод:

```
for a_device in all_devices:
   ...:     net_connect = ConnectHandler(**a_device)
   ...:     output = net_connect.send_command("show arp")
   ...:     print(f"\n\n--------- Device {a_device["device_type"]} ---------")
   ...:     print(output)
   ...:     print("--------- End ---------")
```

В примере будет запрошена таблица ARP:

```
--------- Device cisco_ios ---------
Protocol  Address          Age (min)  Hardware Addr   Type   Interface
Internet  10.10.10.1            32   0062.ec29.70fe  ARPA   GigabitEthernet0/0/0
Internet  10.10.10.20          156   c89c.1dea.0eb6  ARPA   GigabitEthernet0/0/0
Internet  10.10.10.22            -   a093.5141.b780  ARPA   GigabitEthernet0/0/0
Internet  10.10.10.37          178   0001.00ff.0001  ARPA   GigabitEthernet0/0/0
Internet  10.10.10.38           15   0002.00ff.0001  ARPA   GigabitEthernet0/0/0
--------- End ---------


--------- Device cisco_asa ---------
        outside 10.10.10.1 0062.ec29.70fe 1949
        outside 10.10.10.20 c89c.1dea.0eb6 10226
        outside 10.10.10.37 0001.00ff.0001 11606
        outside 10.10.10.38 0002.00ff.0001 11636

--------- End ---------


--------- Device cisco_xr ---------

Thu Jan  3 00:45:44.240 UTC

-------------------------------------------------------------------------------
0/0/CPU0
-------------------------------------------------------------------------------
Address         Age        Hardware Addr   State      Type  Interface
10.10.10.1      00:32:36   0062.ec29.70fe  Dynamic    ARPA  GigabitEthernet0/0/0/0
10.10.10.19     02:50:35   0024.c4e9.48ae  Dynamic    ARPA  GigabitEthernet0/0/0/0
10.10.10.20     00:30:26   c89c.1dea.0eb6  Dynamic    ARPA  GigabitEthernet0/0/0/0
10.10.10.21     03:09:20   1c6a.7aaf.576c  Dynamic    ARPA  GigabitEthernet0/0/0/0
10.10.10.22     02:58:54   a093.5141.b780  Dynamic    ARPA  GigabitEthernet0/0/0/0
10.10.10.23     01:37:37   502f.a8b1.6900  Dynamic    ARPA  GigabitEthernet0/0/0/0
10.10.10.32     02:56:30   5254.abc7.26aa  Dynamic    ARPA  GigabitEthernet0/0/0/0
10.10.10.37     -          0001.00ff.0001  Interface  ARPA  GigabitEthernet0/0/0/0
10.10.10.38     01:55:33   0002.00ff.0001  Dynamic    ARPA  GigabitEthernet0/0/0/0
10.10.10.39     00:11:32   6464.9be8.08c8  Dynamic    ARPA  GigabitEthernet0/0/0/0
10.10.10.42     00:00:46   ec38.739e.2f08  Dynamic    ARPA  GigabitEthernet0/0/0/0
10.10.10.43     00:22:35   5254.abda.5495  Dynamic    ARPA  GigabitEthernet0/0/0/0
--------- End ---------
```

Задание настроек конфигурации

В завершение сегодняшней статьи рассмотрим еще одну задачи по формированию списка VLAN на 3 сетевых коммутаторах:

"Формирование словарей с данными о сетевых устройствах"

```
iosv_l2_s1 = {
```

```
    "device_type": "cisco_ios",
    "ip": "192.168.122.72",
    "username": "david",
    "password": "cisco"
}
iosv_l2_s2 = {
    "device_type": "cisco_ios",
    "ip": "192.168.122.82",
    "username": "david",
    "password": "cisco"
}
iosv_l2_s3 = {
    "device_type": "cisco_ios",
    "ip": "192.168.122.83",
    "username": "david",
    "password": "cisco"
}
all_devices = [iosv_l2_s1, iosv_l2_s2, iosv_l2_s3]# Запись словарей в список
```

"Запуск цикла для последовательного подключения к оборудованию"

```
for devices in all_devices:
    net_connect = ConnectHandler(**devices) # считывание пары ключ-значение
    for n in range (2,21):  # внутренний цикл от 2 до 21
        print ("Creating VLAN " + str(n))  # вывод на экран строки о создание VLAN с номером
        config_commands = ["vlan " + str(n), "name Python_VLAN " + str(n)]   # формирование команды для отправки
        output = net_connect.send_config_set(config_commands)# отправка команды на устройство
        print (output) # вывод на экран переданной конфигурации
```

Небольшая шпаргалка по методам взаимодействия с оборудованием:

- net_connect.send_command() - отправка единичной команды, по умолчанию возвращает вывод;
- net_connect.send_command_timing() - отправка команды по таймингу, по умолчанию возвращает вывод;
- net_connect.send_config_set() - отправка списка команд;
- net_connect.send_config_from_file() - отправка конфигурации из указанного файла;
- net_connect.save_config() - сохранить текущую конфигурацию в режиме startup-config;
- net_connect.enable() - Вход в enable mode;
- net_connect.disconnect() - Закрытие соединения.

Вместо заключения

Сегодня мы начали знакомиться с многофункциональной библиотекой Netmiko, разобрали некоторые практические задачи, которые вы уже сможете выполнить самостоятельно по аналогии с примерами. Планируется 2 статья по этой библиотеке, где уже будет больше практических примеров, позволяющих еще больше автоматизировать ваши рутинные задачи.