Автоматизация работы сетевого администратора, часть 2. Практические кейсы с Telnetlib

Приветствую читателей портала NAG.RU!

Мы продолжаем погружаться в мир автоматизации работы с сетевым оборудованием. В первой части познакомились с теоретическими основами, это лишь необходимый минимум, вам потребуется дополнительно самостоятельно изучать материалы, возможно мы еще этого коснемся в следующих статьях.

Сегодня же мы продолжим знакомиться с инструментами, в частности с модулем ЯП (язык программирования) Python - Telnetlib. Поговорим о задачах с множественным созданием объектов, затронем процедуру бэкапа конфигурации.

Оффтоп: Отмечу активное обсуждение первой статьи, спасибо за ваши мнения и мысли, особенно если они по делу.

Цель нашего цикла - показать инструменты и рассказать о технологиях в мире автоматизации работы сетевого администратора. Учитывая большое количество вендоров в конечной инфраструктуре, мы намеренно используем средства, которые так или иначе унифицированы и подойдут большинству из вас.

Статьи рассчитаны для людей с небольшим бэкграундом работы, либо тех кто давно занимается сетевым администрированием, но только изучает процессы автоматизации. Поэтому наберитесь терпения, мы начинаем с некоторых азов, которые полезно знать каждому из нас.

Кейс по созданию VLAN

Итак, вы сетевой инженер или администратор сети, которому поставили задачу сконфигурировать на L2-коммутаторе некоторое количество VLAN-интерфейсов. Допустим, для 2-4 интерфейсов вы быстро отправите команды в CLI (либо через другую оболочку), но что делать если нужно создать 10,50,100 VLAN ?

Конечно возможно подготовить заранее сконфигурированный файл-шаблон и скопировать его терминал оборудования, но это не так интересно, как написать скрипт, который сделает это автоматически.

Напомню, что мы познакомились уже с Python-библиотекой TelnetLib, она способна подключаться к оборудованию с помощью одноименного протокола Telnet и передавать команды. Пришло время применить наши знания, но вначале кратко о макете:

Виртуальное оборудование настроено в достаточно известном эмуляторе GNS3 и включает в себя:

1. Host-машина в виде Docker Container - **"Network Automation"**. Этот предустановленный объект есть в GNS3, в нем содержатся необходимые пакеты для автоматизации, в том числе Python3.

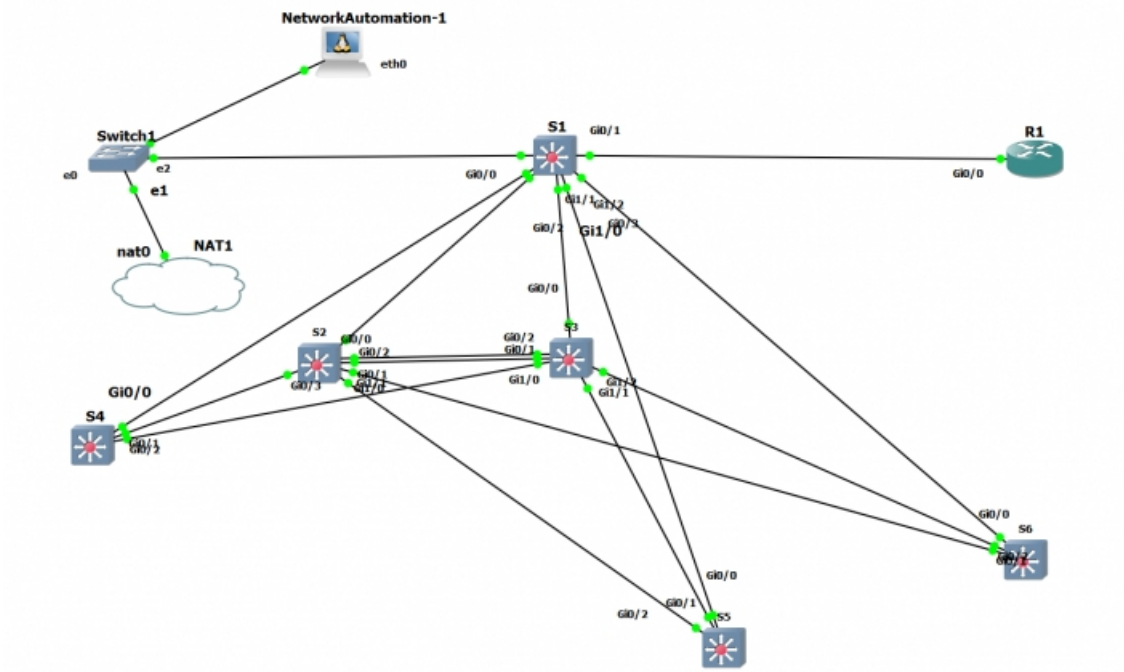
2. **L2-коммутатор** - "S(1-6)". На макете используется оборудование Cisco, единственная проблема это образ ОС - IOS (15.2), он распространяется платно, но вы можете установить оборудование и других вендоров.

- **General**
  - Template name: Cisco IOSvL2 15.2(4.0.55)E
  - Template ID: 024cd476-5eb3-4e2c-9b8c-fcbbce8ce6b2
  - Default name format: {name}-{0}
  - Server: GNS3 VM (GNS3 VM)
  - Console type: telnet
  - Auto start console: False
  - CPUs: 1
  - Memory: 756 MB
  - Linked base VM: True
  - QEMU binary: qemu-system-x86_64
- **Hard disks**
  - Disk image (hda): vios_l2-adventerprisek9-m.vmdk.SSA.152-4.(
  - Disk interface (hda): virtio
- **Network**
  - Adapters: 16
  - Name format: Gi{1}/{0}
  - Segment size: 4
  - Type: e1000
- **Optimizations**
  - CPU throttling: disabled
  - Process priority: normal
- **Additional options**
  - On close: power_off

3. **L3 маршрутизатор** - "R1" . Также эмулируется оборудование Cisco с IOS (15.6).

- **General**
  - Template name: Cisco IOSv 15.6(2)T
  - Template ID: 41942d63-35ac-46ff-95a7-2c3b408250d2
  - Default name format: {name}-{0}
  - Server: GNS3 VM (GNS3 VM)
  - Console type: telnet
  - Auto start console: False
  - CPUs: 1
  - Memory: 512 MB
  - Linked base VM: True
  - QEMU binary: qemu-system-x86_64
- **Hard disks**
  - Disk image (hda): vios-adventerprisek9-m.vmdk.SPA.156-2.T
  - Disk interface (hda): virtio
  - Disk image (hdb): IOSv_startup_config.img
  - Disk interface (hdb): virtio
- **Network**
  - Adapters: 4
  - Name format: Gi0/{0}
  - Type: e1000
- **Optimizations**
  - CPU throttling: disabled
  - Process priority: normal
- **Additional options**
  - On close: power_off

Сама схема топологии сети представлена ниже:



Для начала разберем более простой скрипт, как обычно после символа # будет следовать комментарий к строке, файл написан в формате ".py"

```
import getpass # импорт библиотеки getpass
import telnetlib # импорт библиотеки telnetlib

HOST = "192.168.122.72" # ip-адрес коммутатора Cisco
user = input("Enter your telnet username: ") # запрос ввода логина
password = getpass.getpass() # запрос ввода пароля в зашифрованном виде

tn = telnetlib.Telnet(HOST) # инициализация подключения telnet

tn.read_until(b"Username: ") # ожидание строки Username от CLI
tn.write(user.encode("ascii") + b"\n") #передача данных пользователя и ENTER
```

"ЕСЛИ требуется пароль, то передать"

```
if password:
    tn.read_until(b"Password: ")
    tn.write(password.encode("ascii") + b"\n")

tn.write(b"enable\n") # вход в режим enable
tn.write(b"cisco\n")   # ввод пароля для enable, не всегда требуется
tn.write(b"conf t\n")  # переход в конфигурационный режим
```

"Создание VLAN"

```
tn.write(b"vlan 2\n")
tn.write(b"name Python_VLAN_2\n")
tn.write(b"vlan 3\n")
tn.write(b"name Python_VLAN_3\n")
tn.write(b"vlan 4\n")

print(tn.read_all().decode("ascii")) # вывести на экран команды и результат
```

И все вроде бы неплохо в приложенном скрипте (он даже работает), кроме того что мы конфигурируем лишь 1 коммутатор и вручную прописываем каждый VLAN, хотя меняется лишь его номер. То есть фактически задача по автоматизации конфигурирования большого числа VLAN не выполнена. Пришло время обратиться к циклам, сегодня говорим о **FOR**.

Краткая теория для цикла FOR

Очень часто одно и то же действие необходимо выполнить для набора однотипных данных. Например, преобразовать все строки в списке в верхний регистр. Для выполнения таких действий в Python используется **цикл FOR**.

FOR перебирает по одному элементу из указанной последовательности и совершает над ним определенные действия, которые вы укажите .

```
In [10]: words = ['list', 'dict', 'tuple']

In [11]: upper_words = []

In [12]: for word in words:
    ...:     upper_words.append(word.upper())
    ...:

In [13]: upper_words
Out[13]: ['LIST', 'DICT', 'TUPLE']
```

Кратко о том что происходит на картинке выше:

1. Создается переменная words (тип список) и в нее записываем 3 строки.
2. Создается переменная upper_words (пустой список) .
3. Цикл FOR перебирает каждый элемент из списка words
4. В переменную upper_words добавляется каждый элемент из списка в words - append, при этом применяется метод upper (преобразование в верхний регистр).
5. Выводим на экран переменную upper_words и видим что цикл прошелся по всему списку words, преобразовал каждую строку в верхний регистр и записал их в новый список (upper_words).

В рамках FOR нам понадобится использовать функцию - range, она позволяет перебирать заданный диапазон чисел, выглядит это так:

```
for n in range (2,101):
    print ("vlan" + str(n))
```

Вывод на экран:

```
vlan2
vlan3
vlan4
....
vlan100
```

Важное замечание: переменная n изначально идет как число (тип integer), поэтому при выводе на экран мы ее преобразуем в строку (тип str).

Тем, кому до сих пор осталось ничего не понятно, рекомендую почитать теорию о циклах: 1 статья, 2 статья и т.п.

Мы же вернемся к задаче конфигурирования большого числа VLAN на L2-коммутаторе и перепишем первичный скрипт с учетом знакомства с циклом FOR и функцией range:

```
import getpass # импорт библиотеки getpass
import telnetlib  # импорт библиотеки telnetlib

HOST = "192.168.122.72" # ip-адрес маршрутизатора Cisco
user = input("Enter your telnet username: ") # запрос ввода имени
password = getpass.getpass()  # запрос ввода пароля в зашифрованном виде

tn = telnetlib.Telnet(HOST) # инициализация подключения telnet

tn.read_until(b"Username: ") # ожидание строки Username от CLI
tn.write(user.encode("ascii") + b"\n") # передача данных пользователя и ENTER

"ЕСЛИ требуется пароль, то передать"

if password:
    tn.read_until(b"Password: ")
    tn.write(password.encode("ascii") + b"\n")

tn.write(b"enable\n") # вход в режим enable
tn.write(b"cisco\n")   # ввод пароля для enable, не всегда требуется
tn.write(b"conf t\n")  # переход в конфигурационный режим
```

"Запуск цикла с перебором переменной n от 2 до 101"

```
for n in range (2,101):
    tn.write(b"vlan " + str(n).encode("ascii") + b"\n") # вход в режим enable
    tn.write(b"name Python_VLAN_" + str(n).encode("ascii") + b"\n") # отправка команды в виде строки с подстановкой числа n согласно шагу цикла

tn.write(b"end\n") # выход из режима конфигурации
tn.write(b"wr\n") # запись конфигурации
tn.write(b"exit\n") # выход из подключения к коммутатору
print(tn.read_all().decode("ascii")) # вывести на экран команды и результат
```

Итак, скрипт идентичен с первым, но мы добавили цикл FOR и функцию range, они позволяют избавиться от строчек с ручной передачей конкретного VLAN. Теперь задается диапазон необходимых значений, на основе которых меняется строка типа: vlan№ , в которую подставляется переменная "n".

Кейс по созданию бекапа на L2-коммутаторах

Мы уже умеем использовать цикл FOR для задания VLAN, в следующем практическом примере попробуем автоматизировать процесс работы с несколькими L2-коммутаторами и соберем копию конфигурацию на хостовую машину:

Скрипт будет использовать файл "myswitches" в нем содержатся IP-адреса L2-коммутаторов (S1-S6).

```
import getpass # импорт библиотеки getpass
import telnetlib # импорт библиотеки telnetlib

user = input("Enter your telnet username: ") # запрос ввода имени
password = getpass.getpass() # запрос ввода пароля в зашифрованном виде

f = open("myswitches") # открытие файла "myswitches"
```

"Запуск цикла с перебором IP-адреса (из файла myswitches)"

```
for IP in f:
    IP=IP.strip() # удаляет все пробелы из строки с IP-адресом
    print ("Get running config from Switch " + (IP)) # вывод информации на экран с текущим IP

    HOST = IP # передача IP-адреса в переменную
    tn = telnetlib.Telnet(HOST)     # подключение к L2-коммутатору
    tn.read_until(b"Username: ")  # ожидание строки Username от CLI
    tn.write(user.encode("ascii") + b"\n") # передача данных пользователя и ENTER
```

"ЕСЛИ требуется пароль, то передать"

```
    if password:
        tn.read_until(b"Password: ")
        tn.write(password.encode("ascii") + b"\n")

    tn.write(b"terminal length 0\n") # увеличить область вывода для терминала
    tn.write(b"show run\n") "      # текущая конфигурация
    tn.write(b"exit\n")                  # выйти

    readoutput = tn.read_all()   # записать весь вывод терминала
    saveoutput =  open("switch" + HOST, "w")  # создать файл с именем: switch + IP-адрес

    saveoutput.write(readoutput.decode("ascii")) # передать вывод терминала в файл
    saveoutput.write("\n") # перевести курсор на новую строку
    saveoutput.close()     # закрыть файл
```

Итак, получился скрипт, который берет IP-адрес из заранее подготовленного файла, выполняет подключение до устройства, выводит на экран текущую конфигурацию, а после ее сохраняет в новый файл. Соответственно решили задачу по множественному подключению к L2-коммутатору и собрали бекап файлы.

Вместо заключения

Сегодня мы продолжили знакомиться с реализацией Python-библиотеки Telnetlib, подробно

рассмотрели два практических кейса, в которых познакомились с циклом FOR и некоторыми функциями из языка Python. Вы уже способны написать свой первый скрипт и начать процесс автоматизации вашей работы, у нас же впереди новые инструменты и примеры с ними, читайте NAG.RU!