Автоматизация 7. Napalm в действии

Доброго времени суток, читатели портала NAG.RU и все те, кто неравнодушен к автоматизации процессов сетевого администрирования. В предыдущей статье из цикла (№6) мы познакомились с мощной библиотекой NAPALM, рассмотрели базовые команды и встроенные методы. Сегодня мы изучим готовые скрипты, т.е. закрепим пройденную теорию на практических примерах. Как обычно, ниже будут ссылки на предыдущие части цикла:

1. Автоматизация работы сетевого администратора
2. Автоматизация работы сетевого администратора, часть 2. Практические кейсы с Telnetlib
3. Автоматизация работы сетевого администратора. Часть 3. Знакомство с Paramiko
4. Netmiko основы
5. Netmiko на практике
6. Napalm основы

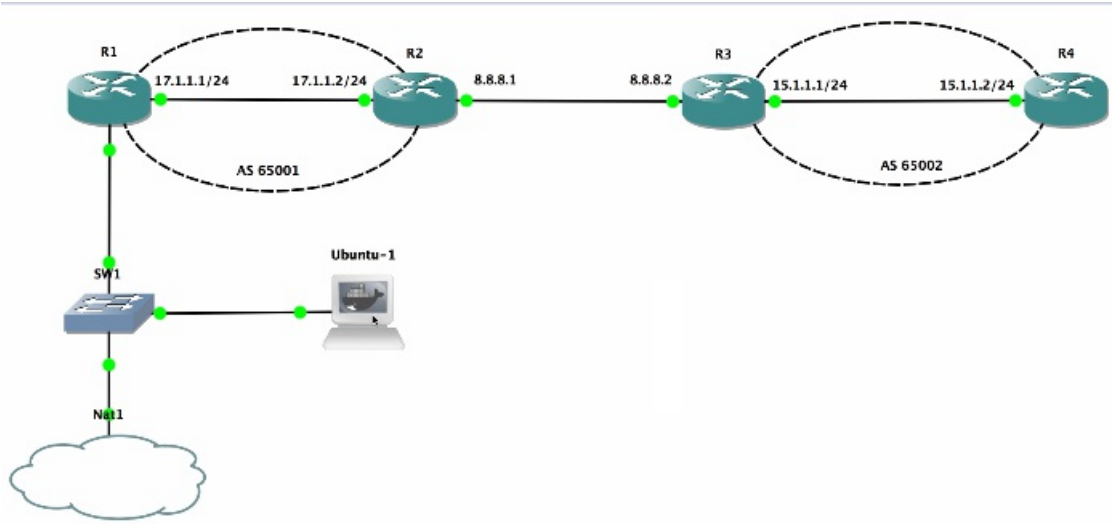
Работа с BGP

Поскольку библиотека поддерживает большое количество встроенных методов, то порой нет необходимости вспоминать legacy команду для сбора данных в CLI. Достаточно лишь обратиться по ссылке и выбрать нужный метод.

В данном примере мы рассмотрим "get_bgp_neighbors", который позволит нам получать информацию о работе протокола BGP на маршрутизирующих устройствах.

Первоначальная топология:

По 2 маршрутизатора в каждой из AS, управляющий хост на Ubuntu (Python3), L2-коммутатор, Интернет.



Задача:

Собирать информацию о разных AS в рамках работы BGP.

Скрипт:

```
"Импорт библиотек"
import json
from napalm import get_network_driver
"Формирование списка из IP-адресов опрашиваемых устройств"
bgplist = ["17.1.1.1",
        "17.1.1.2",
        "8.8.8.2",
        "15.1.1.2"
        ]
"Запуск цикла последовательного сбора данных"
for ip_address in bgplist:
    print ("Connecting to " + str(ip_address))
```

```
# указание драйвера
driver = get_network_driver("ios")

# передача аргументов для подключения
iosv_router = driver(ip_address, "david", "cisco")

# открытие соединения
iosv_router.open()

# сбор данных BGP
bgp_neighbors = iosv_router.get_bgp_neighbors()

# вывод на экран в формате JSON и закрытие сессии
print (json.dumps(bgp_neighbors, indent=4))
iosv_router.close()
```

Вывод на экран:

```
root@Ubuntu-1:~# python bgp.py
{
    "global": {
        "router_id": "17.1.1.2",
        "peers": {
            "17.1.1.1": {
                "is_enabled": true,
                "uptime": 1620,
                "remote_as": 65001,
                "description": "",
                "remote_id": "17.1.1.1",
                "local_as": 65001,
                "is_up": true,
                "address_family": {
                    "ipv4": {
                        "sent_prefixes": 4,
                        "accepted_prefixes": 2,
                        "received_prefixes": 2
                    }
                }
            },
```

Соответственно, для каждого из маршрутизаторов, указанного в списке "bgplist", будут собраны данные о его конфигурации по протоколу BGP.

Работа с ACL

ACL (Access Control List) - набор текстовых выражений, который либо что-то разрешает, либо запрещает. На сетевом оборудовании активно используется ACL для организации firewall-политик, где как правило указываются Source / Destination в виде IP-адреса или сетевых портов.

Для начала создадим отдельный файл, где напишем ACL правила для трафика (ACL1.txt):

```
access-list 100 permit icmp any any
access-list 100 permit tcp any any eq domain
access-list 100 permit tcp any any eq www
access-list 100 permit tcp any any eq 443
```

Кратко то что создали: ACL под номером 100 разрешает ICMP, DNS, WEB-трафик(80/443 порт).

Далее напишем скрипт, который будет сравнивать ACL-кандидат с текущим ACL на оборудование:

```
"Импорт библиотек"
import json
from napalm import get_network_driver

# указание драйвера и инициализация подключения
driver = get_network_driver("ios")
iosvl2 = driver("192.168.122.72", "david", "cisco")
iosvl2.open()

# Проверка различий в кандидате и текущем файле
iosvl2.load_merge_candidate(filename="ACL1.cfg")

# Подтверждение коммита
iosvl2.commit_config()

# Закрытие соединения
iosvl2.close()
```

Таким образом, такой небольшой скрипт позволяет загрузить ACL-политику на устройство, вывести на экран отличия от текущих настроек (если они имеются).

Конечно же хочется всегда получать ожидаемый результат от выполнения программы, что будет если отправлять конфигурацию, которая уже загружена? Или добавлять лишь некоторые строки к ней? Для ответа на вопрос модернизируем наш первоначальный скрипт:

```
import json
from napalm import get_network_driver
driver = get_network_driver("ios")
iosvl2 = driver("192.168.122.72", "david", "cisco")
iosvl2.open()

print ("Accessing 192.168.122.72")
iosvl2.load_merge_candidate(filename="ACL1.cfg")

diffs = iosvl2.compare_config()

# Сравнение с текущим конфигом и соответствующие действия
if len(diffs) > 0:
    print(diffs)
    iosvl2.commit_config()
else:
    print("No changes required.")
    iosvl2.discard_config()

iosvl2.close()
```

Теперь в случае отсутствия изменений в применяемых настройках будет получено следующее:

```
root@Ubuntu-1:~# cat ACL1.cfg
access-list 100 permit icmp any any
access-list 100 permit tcp any any eq domain
access-list 100 permit tcp any any eq www
access-list 100 permit tcp any any eq 443

root@Ubuntu-1:~# python napalmconfig2.py
Accessing 192.168.122.72
No changes required.
```

Если же существовали какие-либо отличия, то перед их применением они выводятся на экран:

```
root@Ubuntu-1:~# python napalmconfig2.py
Accessing 192.168.122.72
+access-list 100 permit icmp any any
+access-list 100 permit tcp any any eq domain
+access-list 100 permit tcp any any eq www
+access-list 100 permit tcp any any eq 443
```

Работа с конфигурацией

В конце разберем полноценный скрипт, который написан с использованием функции отправки

конфигурации и передачи аргументов при запуске кода.

```
# Импорт библиотек

import napalm
import sys
import os

# Функция по конфигурированию файла
def main(config_file):
    """Load a config for the device."""

    # Проверка на наличие переданного пути для файла-кандидата
    if not (os.path.exists(config_file) and os.path.isfile(config_file)):
        # Формирование сообщения об ошибке в виде отсутствия файла
        msg = "Missing {0}".format(config_file)
        raise ValueError(msg)

    # Вывод на экран информации о загружаемом файле настроек
    print("Loading config file {0}.".format(config_file))

    # Использование драйвера для подключения:
    driver = napalm.get_network_driver("eos")

    # Передача параметров для подключения:
    device = driver(
        hostname="127.0.0.1",
        username="vagrant",
        password="vagrant",
        optional_args={"port": 12443},
    )

    # Инициализация подключения
    print("Opening ...")
    device.open()

    print("Loading replacement candidate ...")
    device.load_replace_candidate(filename=config_file)

    # Вывод на экран различий в текущей и загружаемой конфигурации
    # the configuration you can check the changes:
    print("\nDiff:")
    print(device.compare_config())

    # Опрос о решение принять конфигурацию
    try:
        choice = raw_input("\nWould you like to commit [yN]: ")
    except NameError:
        choice = input("\nWould you like to commit? [yN]: ")
    if choice == "y":
        print("Committing ...")
        device.commit_config()
    else:
        print("Discarding ...")
        device.discard_config()

    # Закрыть сессию с устройством.
    device.close()
    print("Done.")

" Запуск основного тела программы с проверкой переданного аргумента и вызова функции config file"
if __name__ == "__main__":
    if len(sys.argv) < 2:
        print("Please supply the full path to "new_good.conf"")
        sys.exit(1)
    config_file = sys.argv[1]
    main(config_file)
```

Вместо заключения

Таким образом, в данной статье разобрали 3 практических примера, которые на самом деле являются только вершиной айсберга, если речь идет о библиотеке Napalm. Ниже будут приведены полезные ссылки, способные помочь при самостоятельном изучении:

- Документация;
- Примеры;
- Статья Cisco о возможностях;
- Лабораторные от Cisco.

На данном этапе цикл статей об автоматизации работы с сетевым оборудованием завершен, возможно еще будут бонусные статьи об интересных библиотеках. В конце, предлагаю читателям проголосовать об интересных для них темах, которые могут быть описаны в будущих циклах:

- Изучение компьютерных сетей с практикой;
- Изучение Python для сетевого инженера;
- REST API;
- Изучение Ansible для сетевого инженера.