Автоматизируем работу с коммутаторами SNR при помощи ANSIBLE

Тема автоматизации управления сетевым оборудованием активно развивается последние несколько лет в связи с ростом количества устройств и предоставляемых сервисов. Мы решили не обходить эту тему стороной и рассказать о том, как можно автоматизировать управление коммутаторами SNR при помощи ansible.

Ansible – один из наиболее популярных инструментов автоматизации, изначально созданный для управления конфигурациями серверов. Однако, сейчас он широко используется и для автоматизации управления сетевого оборудования. Я не буду подробно рассказывать о работе с ansible, в сети достаточно материалов на эту тему, например цикл видеоуроков по ansible на Youtube, или пособие по Ansible для сетевых инженеров от Натальи Самойленко.

Тем не менее, я расскажу базовые принципы работы ansible.

Ansible работает без установки агента на управляемые хосты и отправляет команды по SSH.

Список устройств, к которым ansible будет подключаться, хранится в Inventory или инвентарном файле. В нем хранятся сгруппированные IP-адреса устройств и переменные.

Playbook (файл сценариев) - это файл, в котором описываются действия, которые нужно выполнить на определенной группе хостов.

В качестве примера создадим сценарий для обновления ПО на коммутаторах SNR, используя ansible версии 2.9.6.

Для начала создадим инвентарный файл с именем "invent", в котором укажем IP-адреса коммутаторов и необходимые переменные:

```
[S2985]
192.168.2.4
192.168.2.5

[S2985:vars]
model=SNR-S2985G-24T
lastFW=7.0.3.5(R0241.0339)
FWfilename="SNR-S2985G-48T(24T_8T)(POE)(UPS)(RPS)_7.0.3.5(R0241.0339)_nos.img"
upgradeurl=tftp://192.168.2.1/
FWSize ="13609688"
```

[S2985G] - в квадратных скобках определяется название группы хостов и ниже указываются IP-адреса устройств, принадлежащих данной группе.

[S2985G:vars] - в таком формате определяются переменные, используемые при выполнении сценария над группой хостов.

Так как мы автоматизируем обновление ПО на коммутаторе, в переменных мы указываем модель коммутатора (model), актуальную версию прошивки (lastFW), имя файла для обновления (FWfilename) , URL сервера, на котором находится прошивка (upgradeurl) и размер прошивки (FWSize)

Файл inventory готов, можно переходить к созданию playbook. Playbook описывается в формате YAML и состоит из сценариев (Play) и задач (Tasks). Каждый сценарий состоит из одной или более задач. Итак, начнем создавать наш Playbook в файле UpgradeFW.yml. Для начала, он будет состоять из двух задач - получение вывода команды show version с коммутатора и его отображение.

```
---
- name: Play 1 - upgrade FW on SNR-S2985G switches
  hosts: S2985
  gather_facts: false
  connection: network_cli

  vars:
    ansible_user: admin
    ansible_ssh_pass: admin
    ansible_network_os: ios

  tasks:
    - name: Get show version
      ios_command:
        commands: show version
```

```
    register: sh_ver

  - name: Display show version
    debug:
      msg: "{{sh_ver}}"
```

Любой Playbook должен начинаться с ‘---’, это требования формата YAML. Рассмотрим остальные ключевые слова:

- name - определяет описание нашего сценария, ключ не обязательный, но настоятельно рекомендуемый к использованию.
- hosts - указание группы хостов из inventory файла, к которому будет применен сценарий.
- gather_facts - сбор информации о хосте, поскольку для коммутаторов SNR это не поддерживается, то отключаем.
- connection: network_cli - определяет тип подключения к управляемому устройству, в данном случае CLI over SSH.

Далее идут определения переменных, которые будут использоваться в данном сценарии. В частности, логин, пароль и тип ОС коммутатора. Поскольку CLI на SNR в большей части совпадает с Cisco, указываем ansible_network_os: ios.

Если при логине на коммутатор ansible попадает в непривилегированный режим, то для корректного перехода в привелигерованный, необходимо применить патч (большое спасибо нашему читателю Дмитрию за него) и в секции vars, в начале сценария, добавить переменные для перехода в привилегированный режим.

```
  ansible_become: yes
  ansible_become_method: enable
  ansible_become_password: pass
```

Далее идет описание самой задачи:

- name как и в случае со сценарием служит для описания выполняемых действий.
- ios_command - модуль для выполнения комманд на оборудовании Cisco. Вследствие близкого синтаксиса он работает на коммутаторах SNR и имеет несколько полезных "фишек", которые я покажу далее.
- commands: show version - собственно команда, которая выполняется на коммутаторе, в нашем случае "show version".
- register - сохраняет результат выполнения команды в переменную (sh_ver), которую мы в дальнейшем будем использовать для определения версии модели и ПО коммутатора.
- debug - позволяет отобразить информацию в процессе выполнения ansible. В нашем случае мы будем её использовать для того, чтобы вывести содержимое переменной sh_ver(по правилам YAML переменная должна быть заключена в две фигурные скобки), в которую как вы помните мы записали вывод команды show version с коммутатора).

Поскольку соединение с хостом будет происходить по протоколу SSH, на коммутаторе требуется включить ssh-server (ssh-server enable). На ПК, в домашнем каталоге пользователя, от которого будет запускаться ansible, в файл /.ssh/config необходимо добавить строки:

```
host *
    KexAlgorithms +diffie-hellman-group1-sha1
   Ciphers aes128-cbc,3des-cbc,aes192-cbc,aes256-cbc
```

а также, добавить хост в .ssh/known_host .

Запускаем наш первый Playbook, ключ -i указывает на имя inventory файла. Затем, указываем файл со сценарием:

```
ansible-playbook -i invent upgradeFW.yml



PLAY [S2985]
*********************************************************************************************************************

TASK [Get model and version]
*********************************************************************************************************
ok: [192.168.2.4]
```

```
TASK [Display show version]
*********************************************************************************************************
ok: [192.168.2.4] => {
    "msg": {
        "changed": false,
        "failed": false,
        "stdout": [

            "SNR-S2985G-24T Device, Compiled on Jan 15 12:52:13 2020\n  sysLocation Building 57/2,Predelnaya
st, Ekaterinburg, Russia\n  CPU Mac f8:80:82:74:10:11\n  Vlan MAC f8:80:82:74:10:10\n  SoftWare Version
7.0.3.5(R0241.0339)\n  BootRom Version 7.2.16\n  HardWare Version 1.0.1\n  CPLD Version N/A\n  Serial
No.:SW044810F311000066\n  Copyright (C) 2020 NAG LLC\n  All rights reserved\n  Last reboot is warm reset.\n
Uptime is 0 weeks, 0 days, 0 hours, 11 minutes"
        ],
        "stdout_lines": [
            [
                "SNR-S2985G-24T Device, Compiled on Jan 15 12:52:13 2020",
                "  sysLocation Building 57/2,Predelnaya st, Ekaterinburg, Russia",
                "  CPU Mac f8:80:82:74:10:11",
                "  Vlan MAC f8:80:82:74:10:10",
                "  SoftWare Version 7.0.3.5(R0241.0339)",
                "  BootRom Version 7.2.16",
                "  HardWare Version 1.0.1",
                "  CPLD Version N/A",
                "  Serial No.:SW044810F311000066",
                "  Copyright (C) 2020 NAG LLC",
                "  All rights reserved",
                "  Last reboot is warm reset.",
                "  Uptime is 0 weeks, 0 days, 0 hours, 11 minutes"
            ]
        ]
    }
}

PLAY RECAP
*********************************************************************************************************************
192.168.2.4                : ok=2    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

Первый сценарий выполнен успешно. Мы получили и отобразили информацию о коммутаторе, теперь нам надо её разобрать, чтобы выделить модель и текущую версию прошивки. Для этого я буду использовать модуль TextFSM, созданный как раз для разбора вывода CLI сетевых устройств.

Для этого необходимо создать шаблон на основе регулярных выражений. В начале мы определяем переменные с указанием формата, а затем определяем, где взять значения этих переменных. Для проверки шаблонов есть удобный инструмент, который позволяет создать шаблон и проверить его на выводе CLI on-line.

Для парсинга вывода команды show version коммутаторов SNR шаблон будет следующий:

```
Value Model (\S+)
Value Uptime (.*)
Value SoftwareVersion (\S+)
Value BootROMVersion (\S+)
Value CPUMAC (\S+)
Value VlanMAC (\S+)
Value SN (\S+)

Start
  ^\s*${Model} Device
  ^\s*SoftWare.*Version ${SoftwareVersion}
  ^\s*BootRom Version ${BootROMVersion}
  ^\s*Uptime is ${Uptime}
  ^\s*CPU Mac ${CPUMAC}
  ^\s*Vlan MAC ${VlanMAC}
  ^\s*Serial No.:${SN}
```

Изменим наш сценарий - вместо вывода команды show version, мы её разберем при помощи textFSM - `sh_ver.stdout[0] | parse_cli_textfsm("sh_ver.textFSM")` (`sh_ver.textFSM` - имя файла с шаблоном для парсинга), и запишем в переменную `switch_facts` при помощи ключевого слова `set_fact`. Затем, чтобы проверить

правильность парсинга, выведем его на экран.

Checking Model и Checking Version служат для того, чтобы остановить выполнение сценария, если модель коммутатора не совпадает с нужной или версия ПО не требует обновления. Делаем это с помощью ключевых слов when (выполняет задачу только если условие true) и fail (останавливает сценарий и выдает сообщение).

Получившийся сценарий:

```
---
- name: Play 1 - upgrade FW on SNR-S2985G switches
  hosts: S2985
  gather_facts: false
  connection: network_cli

  vars:
    ansible_user: admin
    ansible_ssh_pass: admin
    ansible_network_os: ios

  tasks:
   - name: Get model and version
     ios_command:
       commands: show version
     register: sh_ver

   - name: Parse output
     set_fact:
        switch_facts: "{{sh_ver.stdout[0] | parse_cli_textfsm("sh_ver.textFSM") }}"

   - name: Print output
     debug:
       msg: "{{ switch_facts}}"

   - name: Checking Model
     when:
       switch_facts[0]["Model"] != model
     fail:
       msg: Model is not {{model}}

   - name: Checking Version
     when:
       switch_facts[0]["SoftwareVersion"] == lastFW
     fail:
       msg: Version is actual {{lastFW}}
```

Запускаем новый сценарий:

```
 ansible-playbook -i invent upgradeFW.yml

PLAY [S2985] ***********************************************************************************************************************

TASK [Get model and version] *******************************************************************************************************
ok: [192.168.2.4]

TASK [Parse output] ****************************************************************************************************************
ok: [192.168.2.4]

TASK [Print Output] ****************************************************************************************************************
ok: [192.168.2.4] => {
    "msg": [
        {
            "BootROMVersion": "7.2.16",
            "CPUMAC": "f8:80:82:74:10:11",
            "Model": "SNR-S2985G-24T",
            "SN": "SW044810F311000066",
            "SoftwareVersion": "7.0.3.5(R0241.0339)",
            "Uptime": "0 weeks, 0 days, 0 hours, 43 minutes",
            "VlanMAC": "f8:80:82:74:10:10"
```

```
      }
   ]
}

TASK [Checking Model] ***********************************************************************************************************
skipping: [192.168.2.4]

TASK [Checking Version] *********************************************************************************************************
skipping: [192.168.2.4]

PLAY RECAP **********************************************************************************************************************
192.168.2.4                : ok=3    changed=0    unreachable=0    failed=0    skipped=2    rescued=0    ignored=0
```

Как видим, вывод show version разобран корректно, а задачи Checking Model и Checking Version пропущены, поскольку условия в when: не соблюдаются - модель верная и ПО требуется обновить.

Теперь можно приступать к самому обновлению коммутатора. Для этого добавляем в наш сценарий следующие задачи:

```
  - name: Upgrading FW
    ios_command:
      commands:
       - command: "copy {{upgradeurl}}/{{FWfilename}} nos.img"
         prompt: "\[Y/N\]:"
         answer: "y"
    vars:
      ansible_command_timeout: 300
    register: cmd_res
```

Этой задачей мы запускаем копирование файла с прошивкой на коммутатор. Поскольку при перезаписи существующего файла необходимо дать подтверждение, мы используем возможность модуля ios_command - prompt и answer. В prompt указываем строку с вопросом, действительно ли мы хотим переписать файл (достаточно последних символов) , а в answer - ответ.

Поскольку копирование прошивки занимает несколько минут, а стандартный таймаут на выполнение команды ansible - 30 сек., мы увеличиваем его, устанавливая переменную ansible_command_timeout.

И в последней строке параметр register используется для записи результатов выполнения команды в переменную, которую мы будем использовать в дальнейшем для проверки результата обновления.

Следующая задача - проверка результата обновления:

```
  - name: Check upgrade result
    when:
       cmd_res.stdout[0].find("Write ok.") == -1 or
       cmd_res.stdout[0].find("Recv total {{FWSize}} bytes") == -1
    fail:
      msg: Upgrade fail {{ cmd_res.stdout}}
```

При успешной записи файла вывод в CLI выглядит следующим образом:

```
File transfer complete.
Recv total 727952 bytes
Begin to write local file, please wait...


Write ok.
close tftp client.
```

Соответственно, если строк с количеством полученных байт или с успешной записью файла на flash нет, мы прерываем выполнение сценария и выводим лог загрузки файла для его последующего анализа.

Следующими задачами мы сохраняем конфигурацию и перезагружаемся, используя уже знакомый инструмент prompt/answer.

```
  - name: Switch save
    ios_command:
      commands:
        - command: "write"
          prompt: "\[Y/N\]:"
          answer: "y"

  - name: Switch reboot
```

```
    ios_command:
      commands:
        - command: "reload"
          prompt: "\[Y/N\]"
          answer: "y"
    ignore_errors: yes
```

В задачу Switch reboot пришлось добавить строку ignore_errors: yes, при которой игнорируются ошибки при выполнении задачи и сценарий не останавливается, поскольку на некоторых моделях SNR, в частности SNR-S2995, перезагрузка выполняется, но задача завершается с ошибкой по таймауту. Причину такого поведения пока обнаружить не удалось, поэтому ошибку просто игнорируем.

Далее, мы проверяем, что после перезагрузки коммутатор загрузился и версия ПО обновилась:

```
  - name: WAIT FOR SWITCH TO RETURN
    wait_for:
       host: "{{inventory_hostname}}"
       port: 22
       delay: 60
       timeout: 600
    delegate_to: localhost

  - name: Get model and version
    ios_command:
      commands: show version
    register: sh_ver

  - name: Parse output
    set_fact:
      switch_facts: "{{sh_ver.stdout[0] | parse_cli_textfsm("sh_ver.textFSM") }}"

  - name: Checking Version
    debug:
       msg: "{{switch_facts[0]["SoftwareVersion"]}}"
```

Результат выполнения сценария должен быть следующим:

```
ansible-playbook -i invent final.yml
PLAY [S2985] ***************************************************************************************************************
TASK [Get model and version] ***********************************************************************************************
ok: [192.168.2.4]
TASK [Parse output] ********************************************************************************************************
ok: [192.168.2.4]
TASK [Checking Model] ******************************************************************************************************
skipping: [192.168.2.4]
TASK [Checking Version] ****************************************************************************************************
skipping: [192.168.2.4]
TASK [Upgrading FW] ********************************************************************************************************
ok: [192.168.2.4]
TASK [Check upgrade result] ************************************************************************************************
[WARNING]: conditional statements should not include jinja2 templating delimiters such as {{ }} or {% %}. Found:
cmd_res.stdout[0].find("Write ok.") == -1 or cmd_res.stdout[0].find("Recv total {{FWSize}} bytes") == -1
skipping: [192.168.2.4]
TASK [Switch save] *********************************************************************************************************
ok: [192.168.2.4]
TASK [Switch reboot] *******************************************************************************************************
ok: [192.168.2.4]
TASK [WAIT FOR SWITCH TO RETURN] *******************************************************************************************
ok: [192.168.2.4 -> localhost]
TASK [Get model and version] ***********************************************************************************************
ok: [192.168.2.4]
TASK [Parse output] ********************************************************************************************************
ok: [192.168.2.4]
TASK [Checking Version] ****************************************************************************************************
ok: [192.168.2.4] => {
    "msg": "7.0.3.5(R0241.0339)"
}
PLAY RECAP *****************************************************************************************************************
192.168.2.4                : ok=10   changed=0    unreachable=0    failed=0    skipped=3    rescued=0    ignored=0
```

Ansible выдает предупреждение о том, что в задаче Check upgrade result не должны использоваться jinja2 шаблоны, но её можно игнорировать, так как задача выполняется корректно.

На этом наш сценарий завершен. Как видно, ansible достаточно простой в освоении инструмент с широким функционалом. Полные версии playbook, inventory и шаблона textFSM можно скачать тут.

На этом, я надеюсь, цикл статей про автоматизацию SNR при помощи ansible не закончится, в планах написать статью про использование шаблонов Jinja2 совместно с ansible.

Как всегда комментарии и критика приветствуются.