PYTHON ДЛЯ СЕТЕВЫХ ИНЖЕНЕРОВ

ANSIBLE

ANSIBLE

Ansible - это система управления конфигурациями. Ansible позволяет автоматизировать и упростить настройку, обслуживание и развертывание серверов, служб, ПО и др.

Ansible активно развивается в сторону поддержки сетевого оборудования и постоянно появляются новые возможности и модули для работы с сетевым оборудованием.

ANSIBLE

Примеры задач, которые поможет решить Ansible:

- подключение по SSH к устройствам
  - паралелльное подключение к устройствам по SSH
- отправка команд на устройства
- удобный синтаксис описания устройств:
  - можно разбивать устройства на группы и затем отправлять какие-то команды на всю группу
- поддержка шаблонов конфигураций с Jinja2

УСТАНОВКА ANSIBLE

Ansible нужно устанавливать только на той машине, с которой будет выполняться управление устройствами.

Требования к управляющему хосту:

- поддержка Python 3 (тестировалось на 3.6)
- Windows не может быть управляющим хостом

Ansible довольно часто обновляется, поэтому лучше установить его в виртуальном окружении.

УСТАНОВКА ANSIBLE

Установить Ansible можно по-разному.

Так как в книге используется ветка devel, надо установить Ansible таким образом с помощью pip:

```
$ pip install git+git://github.com/ansible/ansible.git@devel
```

ПАРАМЕТРЫ ОБОРУДОВАНИЯ

В примерах раздела используются три маршрутизатора и один коммутатор:

- пользователь: cisco
- пароль: cisco
- пароль на режим enable: cisco
- SSH версии 2
- IP-адреса:
  - R1: 192.168.100.1
  - R2: 192.168.100.2
  - R3: 192.168.100.3
  - SW1: 192.168.100.100

ОСНОВЫ ANSIBLE

ОСНОВЫ ANSIBLE

- Работает без установки агента на управляемые хосты
- Использует SSH для подключения к управляемым хостам
- Выполняет изменения, с помощью модулей Python, которые выполняются на управляемых хостах
- Может выполнять действия локально, на управляющем хосте
- Использует YAML для описания сценариев
- Содержит множество модулей (их количество постоянно растет)
- Легко писать свои модули

ТЕРМИНОЛОГИЯ

- **Control machine** — управляющий хост. Сервер Ansible, с которого происходит управление другими хостами
- **Manage node** — управляемые хосты
- **Inventory** — инвентарный файл. В этом файле описываются хосты, группы хостов. А также могут быть созданы переменные
- **Playbook** — файл сценариев
- **Play** — сценарий (набор задач). Связывает задачи с хостами, для которых эти задачи надо выполнить
- **Task** — задача. Вызывает модуль с указанными параметрами и переменными
- **Module** — модуль Ansible. Реализует определенные функции

QUICK START

Минимум, который нужен для начала работы:

- инвентарный файл - в нем описываются устройства
- изменить конфигурацию Ansible, для работы с сетевым оборудованием
- разобраться с ad-hoc командами - это возможность выполнять простые действия с устройствами из командной строки

ИНВЕНТАРНЫЙ ФАЙЛ

ИНВЕНТАРНЫЙ ФАЙЛ

Инвентарный файл - это файл, в котором описываются устройства, к которым Ansible будет подключаться.

В инвентарном файле устройства могут указываться используя IP-адреса или имена. Устройства могут быть указаны по одному или разбиты на группы.

ИНВЕНТАРНЫЙ ФАЙЛ

Файл описывается в формате INI:

```
r5.example.com
[cisco-routers]
192.168.255.1
192.168.255.2
192.168.255.3
192.168.255.4
[cisco-edge-routers]
192.168.255.1
192.168.255.2
```

ИНВЕНТАРНЫЙ ФАЙЛ

По умолчанию, файл находится в `/etc/ansible/hosts`.

Но можно создавать свой инвентарный файл и использовать его. Для этого нужно, либо указать его при запуске Ansible, используя опцию `-i <путь>`, либо указать файл в конфигурационном файле Ansible.

ИНВЕНТАРНЫЙ ФАЙЛ

Пример инвентарного файла, с использованием нестандартных портов для SSH:

```
[cisco-routers]
192.168.255.1:22022
192.168.255.2:22022
192.168.255.3:22022
[cisco-switches]
192.168.254.1
192.168.254.2
```

Такой вариант указания порта работает только с подключениями OpenSSH и не работает с paramiko.

ИНВЕНТАРНЫЙ ФАЙЛ

Если в группу надо добавить несколько устройств с однотипными именами, можно использовать такой вариант записи:

```
[cisco-routers]
192.168.255.[1-5]
```

В группу попадут устройства с адресами 192.168.255.1-192.168.255.5.

ГРУППА ИЗ ГРУПП

Ansible также позволяет объединять группы устройств в общую группу. Для этого используется специальный синтаксис:

```
[cisco-routers]
192.168.255.1
192.168.255.2
192.168.255.3
[cisco-switches]
192.168.254.1
192.168.254.2
[cisco-devices:children]
cisco-routers
cisco-switches
```

AD HOC КОМАНДЫ

AD HOC КОМАНДЫ

Ad-hoc команды - это возможность запустить какое-то действие Ansible из командной строки.

Такой вариант используется, как правило, в тех случаях, когда надо что-то проверить, например, работу модуля. Или просто выполнить какое-то разовое действие, которое не нужно сохранять.

В любом случае, это простой и быстрый способ начать использовать Ansible.

AD HOC КОМАНДЫ

Сначала нужно создать в локальном каталоге инвентарный файл:

```
[cisco-routers]
192.168.100.1
192.168.100.2
192.168.100.3

[cisco-switches]
192.168.100.100
```

AD HOC КОМАНДЫ

Пример ad-hoc команды:

```
$ ansible cisco-routers -i myhosts -m raw -a "sh ip int br" -u cisco --ask-pass
```

AD HOC КОМАНДЫ

Результат выполнения будет таким:

```
$ ansible cisco-routers -i myhosts -m raw -a "sh ip int br" -u cisco --ask-pass
```

```
SSH password:
192.168.100.1 | FAILED | rc=0 >>
to use the 'ssh' connection type with passwords, you must install the sshpass program

192.168.100.2 | FAILED | rc=0 >>
to use the 'ssh' connection type with passwords, you must install the sshpass program

192.168.100.3 | FAILED | rc=0 >>
to use the 'ssh' connection type with passwords, you must install the sshpass program
```

AD HOC КОМАНДЫ

Ошибка значит, что нужно установить программу sshpass. Эта особенность возникает только когда используется аутентификацию по паролю.

Установка sshpass:

```
$ sudo apt-get install sshpass
```

AD HOC КОМАНДЫ

Команду надо выполнить повторно:

```
$ ansible cisco-routers -i myhosts -m raw -a "sh ip int br" -u cisco --ask-pass
```

Результат выполнения команды

```
SSH password:
192.168.100.1 | SUCCESS | rc=0 >>

Interface                  IP-Address      OK? Method Status                Protocol
Ethernet0/0                192.168.100.1   YES NVRAM  up                    up
Ethernet0/1                192.168.200.1   YES NVRAM  up                    up
Ethernet0/2                unassigned      YES manual administratively down down
Ethernet0/3                unassigned      YES manual up                    up
Loopback0                  10.1.1.1        YES manual up                    up
Shared connection to 192.168.100.1 closed.


192.168.100.2 | SUCCESS | rc=0 >>

Interface                  IP-Address      OK? Method Status                Protocol
Ethernet0/0                192.168.100.2   YES manual up                    up
Ethernet0/1                unassigned      YES unset  administratively down down
Ethernet0/2                192.168.200.1   YES manual administratively down down
Ethernet0/3                unassigned      YES manual up                    up
Loopback0                  10.1.1.1        YES manual up                    up
Connection to 192.168.100.2 closed by remote host.
Shared connection to 192.168.100.2 closed.


192.168.100.3 | SUCCESS | rc=0 >>

Interface                  IP-Address      OK? Method Status                Protocol
Ethernet0/0                192.168.100.3   YES manual up                    up
Ethernet0/1                unassigned      YES unset  administratively down down
Ethernet0/2                192.168.200.1   YES manual administratively down down
Ethernet0/3                unassigned      YES manual up                    up
Loopback0                  10.1.1.1        YES manual up                    up
Loopback10                 10.255.3.3      YES manual up                    up
Shared connection to 192.168.100.3 closed.
```

КОНФИГУРАЦИОННЫЙ ФАЙЛ

КОНФИГУРАЦИОННЫЙ ФАЙЛ

Настройки Ansible можно менять в конфигурационном файле.

Конфигурационный файл Ansible может хранится в разных местах:

- ANSIBLE_CONFIG (переменная окружения)
- ansible.cfg (в текущем каталоге)
- .ansible.cfg (в домашнем каталоге пользователя)
- /etc/ansible/ansible.cfg

Ansible ищет файл конфигурации в указанном порядке и использует первый найденный (конфигурация из разных файлов не совмещается).

КОНФИГУРАЦИОННЫЙ ФАЙЛ

В конфигурационном файле можно менять множество параметров. Полный список параметров и их описание, можно найти в документации.

В текущем каталоге должен быть инвентарный файл myhosts:

```
[cisco-routers]
192.168.100.1
192.168.100.2
192.168.100.3

[cisco-switches]
192.168.100.100
```

КОНФИГУРАЦИОННЫЙ ФАЙЛ

Конфигурационный файл ansible.cfg:

```
[defaults]

inventory = ./myhosts
remote_user = cisco
ask_pass = True
```

КОНФИГУРАЦИОННЫЙ ФАЙЛ

Настройки в конфигурационном файле:

- `[defaults]` - секция описывает общие параметры по умолчанию
- `inventory = ./myhosts` - местоположение инвентарного файла
- `remote_user = cisco` - от имени какого пользователя будет подключаться Ansible
- `ask_pass = True` - этот параметр аналогичен опции --ask-pass в командной строке

КОНФИГУРАЦИОННЫЙ ФАЙЛ

Теперь вызов ad-hoc команды будет выглядеть так:

```
$ ansible cisco-routers -m raw -a "sh ip int br"
```

Теперь не нужно указывать инвентарный файл, пользователя и опцию --ask-pass.

GATHERING

По умолчанию, Ansible собирает факты об устройствах.

Факты - это информация о хостах, к которым подключается Ansible. Эти факты можно использовать в playbook и шаблонах как переменные.

Сбором фактов, по умолчанию, занимается модуль setup.

Но, для сетевого оборудования, модуль setup не подходит, поэтому сбор фактов надо отключить. Это можно сделать в конфигурационном файле Ansible или в playbook.

GATHERING

Для сетевого оборудования нужно использовать отдельные модули для сбора фактов (если они есть).

Отключение сбора фактов в конфигурационном файле:

```
gathering = explicit
```

HOST_KEY_CHECKING

Параметр host_key_checking отвечает за проверку ключей, при подключении по SSH. Если указать в конфигурационном файле `host_key_checking=False`, проверка будет отключена.

Это полезно, когда с управляющего хоста Ansible надо подключиться к большому количеству устройств первый раз.

Чтобы проверить этот функционал, надо удалить сохраненные ключи для устройств Cisco, к которым уже выполнялось подкление. В линукс они находятся в файле ~/.ssh/known_hosts.

HOST_KEY_CHECKING

Если выполнить ad-hoc команду, после удаления ключей, вывод будет таким:

```
$ ansible cisco-routers -m raw -a "sh ip int br"
```

```
SSH password:
192.168.100.1 | FAILED | rc=0 >>
Using a SSH password instead of a key is not possible because Host Key checking is enabled
and sshpass does not support this.  Please add this host's fingerprint to your known_hosts
file to manage this host.

192.168.100.2 | FAILED | rc=0 >>
Using a SSH password instead of a key is not possible because Host Key checking is enabled
and sshpass does not support this.  Please add this host's fingerprint to your known_hosts
file to manage this host.

192.168.100.3 | FAILED | rc=0 >>
Using a SSH password instead of a key is not possible because Host Key checking is enabled
and sshpass does not support this.  Please add this host's fingerprint to your known_hosts
file to manage this host.
```

HOST_KEY_CHECKING

Добавляем в конфигурационный файл параметр host_key_checking:

```
[defaults]

inventory = ./myhosts

remote_user = cisco
ask_pass = True

host_key_checking=False
```

HOST_KEY_CHECKING

И повторим ad-hoc команду:

```
$ ansible cisco-routers -m raw -a "sh ip int br"
```

Результат выполнения команды:

```
SSH password:
192.168.100.1 | SUCCESS | rc=0 >>

Interface                  IP-Address      OK? Method Status                Protocol
Ethernet0/0                192.168.100.1   YES NVRAM  up                    up
Ethernet0/1                192.168.200.1   YES NVRAM  up                    up
Ethernet0/2                unassigned      YES manual administratively down down
Ethernet0/3                unassigned      YES manual up                    up
Tunnel0                    unassigned      YES unset  up                    down
Tunnel1                    unassigned      YES unset  up                    down
Tunnel3                    unassigned      YES unset  up                    down
Tunnel9                    unassigned      YES unset  up                    down
Tunnel10                   unassigned      YES unset  up                    down
Tunnel11                   unassigned      YES unset  up                    down
Tunnel15                   unassigned      YES unset  up                    down
Warning: Permanently added '192.168.100.1' (RSA) to the list of known hosts.
Shared connection to 192.168.100.1 closed.


192.168.100.3 | SUCCESS | rc=0 >>

Interface                  IP-Address      OK? Method Status                Protocol
Ethernet0/0                192.168.100.3   YES manual up                    up
Ethernet0/1                unassigned      YES unset  administratively down down
Ethernet0/2                192.168.200.1   YES manual administratively down down
Ethernet0/3                unassigned      YES manual up                    up
Loopback10                 10.255.3.3      YES manual up                    up
Warning: Permanently added '192.168.100.3' (RSA) to the list of known hosts.
Shared connection to 192.168.100.3 closed.


192.168.100.2 | SUCCESS | rc=0 >>

Interface                  IP-Address      OK? Method Status                Protocol
Ethernet0/0                192.168.100.2   YES manual up                    up
Ethernet0/1                unassigned      YES unset  administratively down down
Ethernet0/2                unassigned      YES manual administratively down down
Ethernet0/3                unassigned      YES manual up                    up
Loopback0                  10.0.0.2        YES manual up                    up
Warning: Permanently added '192.168.100.2' (RSA) to the list of known hosts.
Connection to 192.168.100.2 closed by remote host.
Shared connection to 192.168.100.2 closed.
```

HOST_KEY_CHECKING

Обратите внимание на строки:

```
Warning: Permanently added '192.168.100.1' (RSA) to the list of known hosts.
```

Ansible сам добавил ключи устройств в файл ~/.ssh/known_hosts. При подключении в следующий раз этого сообщения уже не будет.

Другие параметры конфигурационного файла можно посмотреть в документации. Пример конфигурационного файла в репозитории Ansible.

МОДУЛИ ANSIBLE

МОДУЛИ ANSIBLE

Вместе с установкой Ansible устанавливается также большое количество модулей (библиотека модулей). В текущей библиотеке модулей, находится порядка 200 модулей.

Модули отвечают за действия, которые выполняет Ansible. При этом, каждый модуль, как правило, отвечает за свою конкретную и небольшую задачу.

Модули можно выполнять отдельно, в ad-hoc командах или собирать в определенный сценарий (play), а затем в playbook.

МОДУЛИ ANSIBLE

Как правило, при вызове модуля, ему нужно передать аргументы. Какие-то аргументы будут управлять поведением и параметрами модуля, а какие-то передавать, например, команду, которую надо выполнить.

Например, мы уже выполняли ad-hoc команды, используя модуль raw. И передавали ему аргументы:

```
$ ansible cisco-routers -i myhosts -m raw -a "sh ip int br" -u cisco --ask-pass
```

МОДУЛИ ANSIBLE

Выполнение такой же задачи в playbook будет выглядеть так:

```
- name: run sh ip int br
  raw: sh ip int br | ex unass
```

После выполнения, модуль возвращает результаты выполнения в формате JSON.

МОДУЛИ ANSIBLE

Модули Ansible, как правило, идемпотентны. Это означает, что модуль можно выполнять сколько угодно раз, но при этом модуль будет выполнять изменения, только если система не находится в желаемом состоянии.

МОДУЛИ ANSIBLE

В Ansible модули разделены на две категории:

- **core** - это модули, которые всегда устанавливаются вместе с Anible. Их поддерживает основная команда разработчиков Ansible.
- **extra** - это модули на данный момент устанавливаются с Ansible, но нет гарантии, что они и дальше будут устанавливаться с Ansible. Возможно, в будущем, их нужно будет устанавливать отдельно. Большинство этих модулей поддерживаются сообществом.

Также в Ansible модули разделены по функциональности. Список всех категорий находится в документации.

ОСНОВЫ PLAYBOOKS

ОСНОВЫ PLAYBOOKS

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

Внутри playbook:

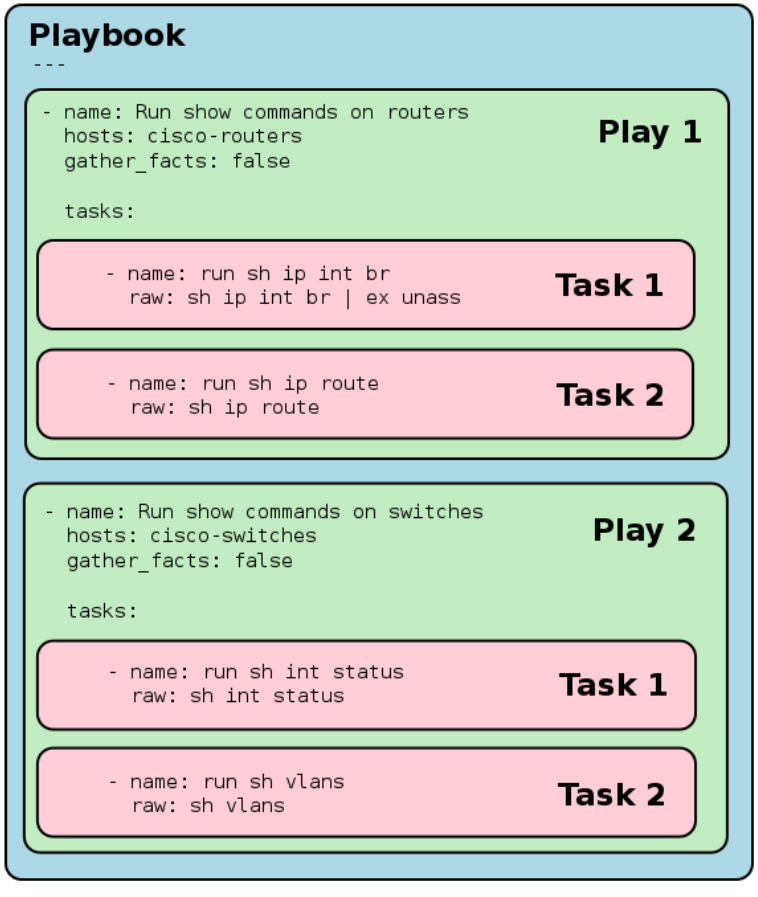
- play - это набор задач, которые нужно выполнить для группы хостов
- task - это конкретная задача. В задаче есть, как минимум:
  - описание (название задачи можно не писать, но очень рекомендуется)
  - модуль и команда (действие в модуле)

СИНТАКСИС PLAYBOOK

Пример plabook 1_show_commands_with_raw.yml:

```
- name: Run show commands on routers
  hosts: cisco-routers
  gather_facts: false

  tasks:

    - name: run sh ip int br
      raw: sh ip int br | ex unass

    - name: run sh ip route
      raw: sh ip route

- name: Run show commands on switches
  hosts: cisco-switches
  gather_facts: false

  tasks:

    - name: run sh int status
      raw: sh int status

    - name: run sh vlan
      raw: show vlan
```

И тот же playbook с отображением элементов:



СИНТАКСИС PLAYBOOK

Так выглядит выполнение playbook:

```
$ ansible-playbook 1_show_commands_with_raw.yml
```

Так выглядит выполнение playbook:

```
PLAY [Run show commands on routers] ********************************************

TASK [run sh ip int br] ********************************************************
changed: [192.168.100.1]
changed: [192.168.100.3]
changed: [192.168.100.2]

TASK [run sh ip route] *********************************************************
changed: [192.168.100.1]
changed: [192.168.100.3]
changed: [192.168.100.2]

PLAY [Run show commands on switches] *******************************************

TASK [run sh int status] *******************************************************
changed: [192.168.100.100]

TASK [run sh vlans] ************************************************************
changed: [192.168.100.100]

PLAY RECAP *********************************************************************
192.168.100.1              : ok=2    changed=2    unreachable=0    failed=0
192.168.100.100            : ok=2    changed=2    unreachable=0    failed=0
192.168.100.2              : ok=2    changed=2    unreachable=0    failed=0
192.168.100.3              : ok=2    changed=2    unreachable=0    failed=0
```

СИНТАКСИС PLAYBOOK

Запуск playbook с опцией -v (вывод сокращен):

```
$ ansible-playbook 1_show_commands_with_raw.yml -v
```

```
SSH password:

PLAY [Run show commands on routers] ********************************************

TASK [run sh ip int br] ********************************************************
changed: [192.168.100.1] => {"changed": true, "rc": 0, "stderr": "Shared connection
 to 192.168.100.1 closed.\r\n", "stdout": "\r\nInterface                  IP-Addres
s      OK? Method Status                Protocol\r\nEthernet0/0                192.
168.100.1   YES NVRAM  up                    up      \r\nEthernet0/1
 192.168.200.1   YES NVRAM  up                    up      \r\nLoopback0
      10.1.1.1        YES manual up                    up      \r\n", "stdout_lines
": ["", "Interface                  IP-Address      OK? Method Status
  Protocol", "Ethernet0/0                192.168.100.1   YES NVRAM  up
       up      ", "Ethernet0/1                192.168.200.1   YES NVRAM  up
            up      ", "Loopback0                10.1.1.1        YES manual up
                 up      "]}
```

ПОРЯДОК ВЫПОЛНЕНИЯ ЗАДАЧ И СЦЕНАРИЕВ

Сценарии (play) и задачи (task) выполняются последовательно, в том порядке, в котором они описаны в playbook.

Если в сценарии, например, две задачи, то сначала первая задача должна быть выполнена для всех устройств, которые указаны в параметре hosts. Только после того, как первая задача была выполнена для всех хостов, начинается выполнение второй задачи.

Если в ходе выполнения playbook, возникла ошибка в задаче на каком-то устройстве, это устройство исключается, и другие задачи на нем выполняться не будут.

ПОРЯДОК ВЫПОЛНЕНИЯ ЗАДАЧ И СЦЕНАРИЕВ

Например, заменим пароль пользователя cisco на cisco123 (правильный cisco) на маршрутизаторе 192.168.100.1, и запустим playbook заново:

```
$ ansible-playbook 1_show_commands_with_raw.yml
```

```
PLAY [Run show commands on routers] ********************************************

TASK [run sh ip int br] ********************************************************
changed: [192.168.100.3]
changed: [192.168.100.2]
fatal: [192.168.100.1]: FAILED! => {"changed": true, "failed": true, "rc": 5, "stderr"
: "", "stdout": "", "stdout_lines": []}

TASK [run sh ip route] *********************************************************
changed: [192.168.100.2]
changed: [192.168.100.3]

PLAY [Run show commands on switches] *******************************************

TASK [run sh int status] *******************************************************
changed: [192.168.100.100]

TASK [run sh vlans] ************************************************************
changed: [192.168.100.100]
        to retry, use: --limit @/home/nata/pyneng_course/chapter15/1_show_commands_wit
h_raw.retry

PLAY RECAP *********************************************************************
192.168.100.1              : ok=0    changed=0    unreachable=0    failed=1
192.168.100.100            : ok=2    changed=2    unreachable=0    failed=0
192.168.100.2              : ok=2    changed=2    unreachable=0    failed=0
192.168.100.3              : ok=2    changed=2    unreachable=0    failed=0
```

ПОРЯДОК ВЫПОЛНЕНИЯ ЗАДАЧ И СЦЕНАРИЕВ

Обратите внимание на ошибку в выполнении первой задачи для маршрутизатора 192.168.100.1.

Во второй задаче 'TASK [run sh ip route]', Ansible уже исключил маршрутизатор и выполняет задачу только для маршрутизаторов 192.168.100.2 и 192.168.100.3.

ПОРЯДОК ВЫПОЛНЕНИЯ ЗАДАЧ И СЦЕНАРИЕВ

Еще один важный аспект - Ansible выдал сообщение:

```
to retry, use: --limit @/home/nata/pyneng_course/chapter15/1_show_commands_with_raw.retry
```

ПОРЯДОК ВЫПОЛНЕНИЯ ЗАДАЧ И СЦЕНАРИЕВ

Если, при выполнении playbook, на каком-то устройстве возникла ошибка, Ansible создает специальный файл, который называется точно так же как playbook, но расширение меняется на retry.

В этом файле хранится имя или адрес устройства на котором возникла ошибка (файл 1_show_commands_with_raw.retry):

```
192.168.100.1
```

ПОРЯДОК ВЫПОЛНЕНИЯ ЗАДАЧ И СЦЕНАРИЕВ

После настройки правильного пароля на маршрутизаторе, перезапускаем playbook:

```
$ ansible-playbook 1_show_commands_with_raw.yml --limit @/home/nata/pyneng_course/chapter1
```

```
SSH password:

PLAY [Run show commands on routers] ********************************************

TASK [run sh ip int br] ********************************************************
changed: [192.168.100.1]

TASK [run sh ip route] *********************************************************
changed: [192.168.100.1]

PLAY RECAP *********************************************************************
192.168.100.1              : ok=2    changed=2    unreachable=0    failed=0
```

ПОРЯДОК ВЫПОЛНЕНИЯ ЗАДАЧ И СЦЕНАРИЕВ

Ansible взял список устройств, которые перечислены в файле retry и выполнил playbook только для них.

Можно запустить playbook и так:

```
$ ansible-playbook 1_show_commands_with_raw.yml --limit @1_show_commands_with_raw.retry
```

ПАРАМЕТР --LIMIT

Параметр --limit позволяет ограничивать, для каких хостов или групп будет выполняться playbook, при этом, не меняя сам playbook.

Например, таким образом playbook можно запустить только для маршрутизатора 192.168.100.1:

```
$ ansible-playbook 1_show_commands_with_raw.yml --limit 192.168.100.1
```

ИДЕМПОТЕНТНОСТЬ

Модули Ansible идемпотентны. Это означает, что модуль можно выполнять сколько угодно раз, но при этом модуль будет выполнять изменения, только если система не находится в желаемом состоянии.

Но, есть исключения из такого поведения. Например, модуль raw всегда вносит изменения. Поэтому в выполнении playbook выше, всегда отображалось состояние changed.

ИДЕМПОТЕНТНОСТЬ

Например, если в задаче указано, что на сервер Linux надо установить пакет httpd, то он будет установлен только в том случае, если его нет. То есть, действие не будет повторяться снова и снова, при каждом запуске. А лишь тогда, когда пакета нет.

Аналогично, и с сетевым оборудованием. Если задача модуля выполнить команду в конфигурационном режиме, а она уже есть на устройстве, модуль не будет вносить изменения.

ПЕРЕМЕННЫЕ

ПЕРЕМЕННЫЕ

Переменной может быть:

- информация об устройстве, которая собрана как факт, а затем используется в шаблоне
- в переменные можно записывать полученный вывод команды
- переменная может быть указана вручную в playbook

ИМЕНА ПЕРЕМЕННЫХ

В Ansible есть определенные ограничения по формату имен переменных:

- Переменные могут состоять из букв, чисел и символа _
- Переменные должны начинаться с буквы

ИМЕНА ПЕРЕМЕННЫХ

Кроме того, можно создавать словари с переменными (в формате YAML):

```
R1:
  IP: 10.1.1.1/24
  DG: 10.1.1.100
```

ИМЕНА ПЕРЕМЕННЫХ

Обращаться к переменным в словаре можно двумя вариантами:

```
R1['IP']
R1.IP
```

При использовании второго варианта, могут быть проблемы, если название ключа совпадает с зарезервированным словом (методом или атрибутом) в Python или Ansible.

ГДЕ МОЖНО ОПРЕДЕЛЯТЬ ПЕРЕМЕННЫЕ

Переменные можно создавать:

- в инвентарном файле
- в playbook
- в специальных файлах для группы/устройства
- в отдельных файлах, которые добавляются в playbook через include (как в Jinja2)
- в ролях, которые затем используются
- можно передавать переменные при вызове playbook

Также можно использовать факты, которые были собраны про устройство, как переменные.

ПЕРЕМЕННЫЕ В ИНВЕНТАРНОМ ФАЙЛЕ

В инвентарном файле можно указывать переменные для группы:

```
[cisco-routers]
192.168.100.1
192.168.100.2
192.168.100.3

[cisco-switches]
192.168.100.100

[cisco-routers:vars]
ntp_server=192.168.255.100
log_server=10.255.100.1
```

ПЕРЕМЕННЫЕ В PLAYBOOK

Переменные можно задавать прямо в playbook

```
- name: Run show commands on routers
  hosts: cisco-routers
  gather_facts: false

  vars:
    ntp_server: 192.168.255.100
    log_server: 10.255.100.1

  tasks:

    - name: run sh ip int br
      raw: sh ip int br | ex unass

    - name: run sh ip route
      raw: sh ip route
```

ПЕРЕМЕННЫЕ В СПЕЦИАЛЬНЫХ ФАЙЛАХ

Ansible позволяет хранить переменные для группы/устройства в специальных файлах:

- Для групп устройств, переменные должны находится в каталоге group_vars, в файлах, которые называются, как имя группы.
  - в каталоге group_vars можно создавать файл all, в котором будут находиться переменные, которые относятся ко всем группам.
- Для конкретных устройств, переменные должны находится в каталоге host_vars, в файлах, которые соответствуют имени или адресу хоста.

ПЕРЕМЕННЫЕ В СПЕЦИАЛЬНЫХ ФАЙЛАХ

Ansible позволяет хранить переменные для группы/устройства в специальных файлах:

- Все файлы с переменными, должны быть в формате YAML. Расширение файла может быть yml, yaml, json или без расширения
- каталоги group_vars и host_vars должны находиться в том же каталоге, что и playbook. Или могут находиться внутри каталога inventory (первый вариант более распространенный).
  - если каталоги и файлы названы правильно и расположены в указанных каталогах, Ansible сам разпознает файлы и будет использовать переменные

ПЕРЕМЕННЫЕ В СПЕЦИАЛЬНЫХ ФАЙЛАХ

Например, если инвентарный файл myhosts выглядит так:

```
[cisco-routers]
192.168.100.1
192.168.100.2
192.168.100.3

[cisco-switches]
192.168.100.100
```

ПЕРЕМЕННЫЕ В СПЕЦИАЛЬНЫХ ФАЙЛАХ

Можно создать такую структуру каталогов:

```
├── group_vars                    _
│   ├── all.yml                    |
│   ├── cisco-routers.yml          |  Каталог с переменными для групп устройств
│   └── cisco-switches.yml        _|
|
├── host_vars                     _
│   ├── 192.168.100.1              |
│   ├── 192.168.100.2              |
│   ├── 192.168.100.3              |  Каталог с переменными для устройств
│   └── 192.168.100.100           _|
|
└── myhosts                        |  Инвентарный файл
```

ПЕРЕМЕННЫЕ В СПЕЦИАЛЬНЫХ ФАЙЛАХ

Файл group_vars/all.yml:

```
cli:
  host: "{{ inventory_hostname }}"
  username: "cisco"
  password: "cisco"
  transport: cli
  authorize: yes
  auth_pass: "cisco"
```

ПЕРЕМЕННЫЕ В СПЕЦИАЛЬНЫХ ФАЙЛАХ

В файле group_vars/all.yml создан словарь cli. В этом словаре перечислены те аргументы, которые должны задаваться для работы с сетевым оборудованием через встроенные модули Ansible

Переменная host: "{{ inventory_hostname }}":

- inventory_hostname - это специальная переменная, которая указывает на тот хост, для которого Ansible выполняет действия.
- синтаксис {{ inventory_hostname }} - это подстановка переменных. Используется формат Jinja

ПЕРЕМЕННЫЕ В СПЕЦИАЛЬНЫХ ФАЙЛАХ

group_vars/cisco-routers.yml

```
log_server: 10.255.100.1
ntp_server: 10.255.100.1
users:
  user1: pass1
  user2: pass2
  user3: pass3
```

ПЕРЕМЕННЫЕ В СПЕЦИАЛЬНЫХ ФАЙЛАХ

group_vars/cisco-switches.yml

```
vlans:
  - 10
  - 20
  - 30
```

ПЕРЕМЕННЫЕ В СПЕЦИАЛЬНЫХ ФАЙЛАХ

Файлы с переменными для хостов однотипны и в них меняются только адреса и имена.

Файл host_vars/192.168.100.1

```
hostname: london_r1
mgmnt_loopback: 100
mgmnt_ip: 10.0.0.1
ospf_ints:
  - 192.168.100.1
  - 10.0.0.1
  - 10.255.1.1
```

ПРИОРИТЕТНОСТЬ ПЕРЕМЕННЫХ

Чаще всего, переменная с определенным именем только одна.

Но, иногда может понадобиться создать переменную в разных местах и тогда нужно понимать, в каком порядке Ansible перезаписывает переменные.

ПРИОРИТЕТНОСТЬ ПЕРЕМЕННЫХ

Приоритет переменных (последние значения переписывают предыдущие):

- Значения переменных в ролях
  - задачи в ролях будут видеть собственные значения. Задачи, которые определены вне роли, будут видеть последние значения переменных роли
- переменные в инвентарном файле
- переменные для группы хостов в инвентарном файле
- переменные для хостов в инвентарном файле

ПРИОРИТЕТНОСТЬ ПЕРЕМЕННЫХ

- переменные в каталоге group_vars
- переменные в каталоге host_vars
- факты хоста
- переменные сценария (play)
- переменные сценария, которые запрашиваются через vars_prompt

ПРИОРИТЕТНОСТЬ ПЕРЕМЕННЫХ

- переменные, которые передаются в сценарий через vars_files
- переменные полученные через параметр register
- set_facts
- переменные из роли и помещенные через include
- переменные блока (переписывают другие значения только для блока)
- переменные задачи (task) (переписывают другие значения только для задачи)
- переменные, которые передаются при вызове playbook через параметр --extra-vars (всегда наиболее приоритетные)

РАБОТА С РЕЗУЛЬТАТАМИ ВЫПОЛНЕНИЯ МОДУЛЯ

VERBOSE

Флаг verbose позволяет подробно посмотреть какие шаги выполняет Ansible.

Пример запуска playbook с флагом verbose (вывод сокращен):

```
ansible-playbook 1_show_commands_with_raw.yml -v
```

```
SSH password:

PLAY [Run show commands on routers] ********************************************

TASK [run sh ip int br] ********************************************************
changed: [192.168.100.1] => {"changed": true, "rc": 0, "stderr": "Shared connection
 to 192.168.100.1 closed.\r\n", "stdout": "\r\nInterface                  IP-Addres
s       OK? Method Status                Protocol\r\nEthernet0/0                192.
168.100.1   YES NVRAM  up                    up      \r\nEthernet0/1
 192.168.200.1   YES NVRAM  up                    up      \r\nLoopback0
      10.1.1.1        YES manual up                    up      \r\n", "stdout_lines
": ["", "Interface                  IP-Address      OK? Method Status
  Protocol", "Ethernet0/0                192.168.100.1   YES NVRAM  up
       up      ", "Ethernet0/1                192.168.200.1   YES NVRAM  up
            up      ", "Loopback0                10.1.1.1        YES manual up
                 up      "]}
```

VERBOSE

При увеличении количества букв v в флаге, вывод становится более подробным.

```
ansible-playbook 1_show_commands_with_raw.yml -vvv
```

VERBOSE

В выводе видны результаты выполнения задачи, они возвращаются в формате JSON:

- **changed** - ключ, который указывает были ли внесены изменения
- **rc** - return code. Это поле появляется в выводе тех модулей, которые выполняют какие-то команды
- **stderr** - ошибки, при выполнении команды. Это поле появляется в выводе тех модулей, которые выполняют какие-то команды
- **stdout** - вывод команды
- **stdout_lines** - вывод в виде списка команд, разбитых построчно

REGISTER

Параметр **register** сохраняет результат выполнения модуля в переменную. Затем эта переменная может использоваться в шаблонах, в принятии решений о ходе сценария или для отображении вывода.

REGISTER

В playbook 2_register_vars.yml, с помощью register, вывод команды sh ip int br сохранен в переменную sh_ip_int_br_result:

```
- name: Run show commands on routers
  hosts: cisco-routers
  gather_facts: false

  tasks:

    - name: run sh ip int br
      raw: sh ip int br | ex unass
      register: sh_ip_int_br_result
```

REGISTER

Если запустить этот playbook, вывод не будет отличаться, так как вывод только записан в переменную, но с переменной не выполяется никаких действий. Следующий шаг - отобразить результат выполнения команды, с помощью модуля debug.

DEBUG

Модуль debug позволяет отображать информацию на стандартный поток вывода. Это может быть произвольная строка, переменная, факты об устройстве.

DEBUG

Для отображения сохраненных результатов выполнения команды, в playbook 2_register_vars.yml добавлена задача с модулем debug:

```
- name: Run show commands on routers
  hosts: cisco-routers
  gather_facts: false

  tasks:

    - name: run sh ip int br
      raw: sh ip int br | ex unass
      register: sh_ip_int_br_result

    - name: Debug registered var
      debug: var=sh_ip_int_br_result.stdout_lines
```

DEBUG

Обратите внимание, что выводится не всё содержимое переменной sh_ip_int_br_result, а только содержимое stdout_lines. В sh_ip_int_br_result.stdout_lines находится список строк, поэтому вывод будут структурированн.

Результат запуска playbook будет выглядит так:

```
$ ansible-playbook 2_register_vars.yml
```

DEBUG

```
SSH password:

PLAY [Run show commands on routers] ********************************************

TASK [run sh ip int br] ********************************************************
changed: [192.168.100.1]
changed: [192.168.100.2]
changed: [192.168.100.3]

TASK [Debug registered var] ****************************************************
ok: [192.168.100.1] => {
    "sh_ip_int_br_result.stdout_lines": [
        "",
        "Interface                  IP-Address      OK? Method Status                Protocol",
        "Ethernet0/0                192.168.100.1   YES NVRAM  up                    up      ",
        "Ethernet0/1                192.168.200.1   YES NVRAM  up                    up      ",
        "Loopback0                  10.1.1.1        YES manual up                    up      "
    ]
}
ok: [192.168.100.2] => {
    "sh_ip_int_br_result.stdout_lines": [
        "",
        "Interface                  IP-Address      OK? Method Status                Protocol",
        "Ethernet0/0                192.168.100.2   YES manual up                    up      ",
        "Ethernet0/2                192.168.200.1   YES manual administratively down down    ",
        "Loopback0                  10.1.1.1        YES manual up                    up      "
    ]
}
ok: [192.168.100.3] => {
    "sh_ip_int_br_result.stdout_lines": [
        "",
        "Interface                  IP-Address      OK? Method Status                Protocol",
        "Ethernet0/0                192.168.100.3   YES manual up                    up      ",
        "Ethernet0/2                192.168.200.1   YES manual administratively down down    ",
        "Loopback0                  10.1.1.1        YES manual up                    up      ",
        "Loopback10                 10.255.3.3      YES manual up                    up      "
    ]
}

PLAY RECAP *********************************************************************
192.168.100.1              : ok=2    changed=1    unreachable=0    failed=0
192.168.100.2              : ok=2    changed=1    unreachable=0    failed=0
192.168.100.3              : ok=2    changed=1    unreachable=0    failed=0
```

REGISTER, DEBUG, WHEN

С помощью ключевого слова **when**, можно указать условие, при выполнении которого, задача выполняется. Если условие не выполняется, то задача пропускается.

when в Ansible используется как if в Python.

REGISTER, DEBUG, WHEN

Пример playbook 3_register_debug_when.yml:

```
- name: Run show commands on routers
  hosts: cisco-routers
  gather_facts: false

  tasks:

    - name: run sh ip int br
      raw: sh ip int bri | ex unass
      register: sh_ip_int_br_result

    - name: Debug registered var
      debug:
        msg: "Error in command"
      when: "'invalid' in sh_ip_int_br_result.stdout"
```

REGISTER, DEBUG, WHEN

Модуль debug отображает не содержимое сохраненной переменной, а сообщение, которое указано в переменной msg.

Задача будет выполнена только в том случае, если в выводе sh_ip_int_br_result.stdout будет найдена строка invalid

```
when: "'invalid' in sh_ip_int_br_result.stdout"
```

Модули, которые работают с сетевым оборудованием, автоматически проверяют ошибки, при выполнении команд. Тут этот пример используется для демонстрации возможностей Ansible.

REGISTER, DEBUG, WHEN

Выполнение playbook:

```
$ ansible-playbook 3_register_debug_when.yml
```

```
SSH password:

PLAY [Run show commands on routers] ********************************************

TASK [run sh ip int br] ********************************************************
changed: [192.168.100.2]
changed: [192.168.100.1]
changed: [192.168.100.3]

TASK [Debug registered var] ****************************************************
skipping: [192.168.100.1]
skipping: [192.168.100.2]
skipping: [192.168.100.3]

PLAY RECAP *********************************************************************
192.168.100.1              : ok=1    changed=1    unreachable=0    failed=0
192.168.100.2              : ok=1    changed=1    unreachable=0    failed=0
192.168.100.3              : ok=1    changed=1    unreachable=0    failed=0
```

REGISTER, DEBUG, WHEN

Выполнение того же playbook, но с ошибкой в команде:

```
- name: Run show commands on routers
  hosts: cisco-routers
  gather_facts: false

  tasks:

    - name: run sh ip int br
      raw: shh ip int bri | ex unass
      register: sh_ip_int_br_result

    - name: Debug registered var
      debug:
        msg: "Error in command"
      when: "'invalid' in sh_ip_int_br_result.stdout"
```

REGISTER, DEBUG, WHEN

Теперь результат выполнения такой:

```
$ ansible-playbook 3_register_debug_when.yml
```

```
SSH password:

PLAY [Run show commands on routers] ********************************************

TASK [run sh ip int br] ********************************************************
changed: [192.168.100.1]
changed: [192.168.100.2]
changed: [192.168.100.3]

TASK [Debug registered var] ****************************************************
ok: [192.168.100.1] => {
    "msg": "Error in command"
}
ok: [192.168.100.2] => {
    "msg": "Error in command"
}
ok: [192.168.100.3] => {
    "msg": "Error in command"
}

PLAY RECAP *********************************************************************
192.168.100.1              : ok=2    changed=1    unreachable=0    failed=0
192.168.100.2              : ok=2    changed=1    unreachable=0    failed=0
192.168.100.3              : ok=2    changed=1    unreachable=0    failed=0
```

МОДУЛИ ДЛЯ РАБОТЫ С СЕТЕВЫМ ОБОРУДОВАНИЕМ

МОДУЛИ ДЛЯ РАБОТЫ С СЕТЕВЫМ ОБОРУДОВАНИЕМ

Модули для работы с сетевым оборудованием, можно разделить на две части:

- модули для оборудования с поддержкой API
- модули для оборудования, которое работает только через CLI

Если оборудование поддерживает API, как например, NXOS, то для него создано большое количество модулей, которые выполняют конкретные действия, по настройке функционала (например, для NXOS создано более 60 модулей).

МОДУЛИ ДЛЯ РАБОТЫ С СЕТЕВЫМ ОБОРУДОВАНИЕМ

Для оборудования, которое работает только через CLI, Ansible поддерживает такие три типа модулей:

- os_command - выполняет команды show
- os_facts - собирает факты об устройствах
- os_config - выполняет команды конфигурации

МОДУЛИ ДЛЯ РАБОТЫ С СЕТЕВЫМ ОБОРУДОВАНИЕМ

Соответственно, для разных операционных систем, будут разные модули. Например, для Cisco IOS, модули будут называться:

- ios_command
- ios_config
- ios_facts

МОДУЛИ ДЛЯ РАБОТЫ С СЕТЕВЫМ ОБОРУДОВАНИЕМ

Аналогичные три модуля доступны для таких ОС:

- Dellos10
- Dellos6
- Dellos9
- EOS
- IOS
- IOS XR
- JUNOS
- SR OS
- VyOS

ВАРИАНТЫ ПОДКЛЮЧЕНИЯ

Ansible поддерживает такие типы подключений:

- **paramiko**
- **SSH** - OpenSSH. Используется по умолчанию
- **local** - действия выполняются локально, на управляющем хосте

При подключении по SSH, по умолчанию используются SSH ключи, но можно переключиться на использование паролей.

ВАРИАНТЫ ПОДКЛЮЧЕНИЯ

По умолчанию, Ansible загружает модуль Python на устройство, для того, чтобы выполнить действия. Если же оборудование не поддерживает Python, как в случае с доступом к сетевому оборудованию через CLI, нужно указать, что модуль должен запускаться локально, на управляющем хосте Ansible.

ОСОБЕННОСТИ ПОДКЛЮЧЕНИЯ К СЕТЕВОМУ ОБОРУДОВАНИЮ

При работе с сетевым оборудованием, есть несколько параметров в playbook, которые нужно менять:

- gather_facts - надо отключить, так как для сетевого оборудования используются свои модули сбора фактов
- connection - управляет тем, как именно будет происходить подключение. Для сетевого оборудования необходимо установить в local

ОСОБЕННОСТИ ПОДКЛЮЧЕНИЯ К СЕТЕВОМУ ОБОРУДОВАНИЮ

То есть, для каждого сценария (play), нужно указывать:

- gather_facts: false
- connection: local

```
- name: Run show commands on routers
  hosts: cisco-routers
  gather_facts: false
  connection: local
```

ОСОБЕННОСТИ ПОДКЛЮЧЕНИЯ К СЕТЕВОМУ ОБОРУДОВАНИЮ

В Ansible переменные можно указывать в разных местах, поэтому те же настройки можно указать по-другому.

Например, в конфигурационном файле:

```
[defaults]

gathering = explicit
```

ОСОБЕННОСТИ ПОДКЛЮЧЕНИЯ К СЕТЕВОМУ ОБОРУДОВАНИЮ

Такой вариант подходит в том случае, когда Ansible используется больше для подключение к сетевым устройствам (или, локальные playbook используются для подключения к сетевому оборудованию).

В таком случае, нужно будет наоборот явно включать сбор фактов, если он нужен.

ОСОБЕННОСТИ ПОДКЛЮЧЕНИЯ К СЕТЕВОМУ ОБОРУДОВАНИЮ

Указать, что нужно использовать локальное подключение, также можно по-разному.

В инвентарном файле:

```
[cisco-routers]
192.168.100.1
192.168.100.2
192.168.100.3

[cisco-switches]
192.168.100.100

[cisco-routers:vars]
ansible_connection=local
```

ОСОБЕННОСТИ ПОДКЛЮЧЕНИЯ К СЕТЕВОМУ ОБОРУДОВАНИЮ

Или в файлах переменных, например, в group_vars/all.yml:

```
ansible_connection: local
```

ОСОБЕННОСТИ ПОДКЛЮЧЕНИЯ К СЕТЕВОМУ ОБОРУДОВАНИЮ

В следующих разделах будет использоваться такой вариант:

```
- name: Run show commands on routers
  hosts: cisco-routers
  gather_facts: false
  connection: local
```

В реальной жизни нужно выбрать тот вариант, который наиболее удобен для работы.

АРГУМЕНТ PROVIDER

Модули, которые используются для работы с сетевым оборудованием, требуют задания нескольких аргументов.

Для каждой задачи должны быть указаны такие аргументы:

- **host** - имя или IP-адрес удаленного устройства
- **port** - к какому порту подключаться
- **username** - имя пользователя
- **password** - пароль
- **transport** - тип подключения: CLI или API. По умолчанию - cli
- **authorize** - нужно ли переходить в привилегированный режим (enable, для Cisco)
- **auth_pass** - пароль для привилегированного режима

АРГУМЕНТ PROVIDER

Ansible также позволяет собрать их в один аргумент - **provider**.

```
tasks:

  - name: run show version
    ios_command:
      commands: show version
      host: "{{ inventory_hostname }}"
      username: cisco
      password: cisco
      transport: cli
```

АРГУМЕНТ PROVIDER

Аргументы созданы как переменная `cli` в playbook, а затем передаются как переменная аргументу provider:

```
vars:
  cli:
    host: "{{ inventory_hostname }}"
    username: cisco
    password: cisco
    transport: cli

tasks:
  - name: run show version
    ios_command:
      commands: show version
      provider: "{{ cli }}"
```

АРГУМЕНТ PROVIDER

И, самый удобный вариант, задавать аргументы в каталоге group_vars.

Например, если у всех устройств одинаковые значения аргументов, можно задать их в файле group_vars/all.yml:

```
cli:
  host: "{{ inventory_hostname }}"
  username: cisco
  password: cisco
  transport: cli
  authorize: yes
  auth_pass: cisco
```

АРГУМЕНТ PROVIDER

Затем переменная используется в playbook так же, как и в случае указания переменных в playbook:

```
tasks:
  - name: run show version
    ios_command:
      commands: show version
      provider: "{{ cli }}"
```

АРГУМЕНТ PROVIDER

Кроме того, Ansible поддерживает задание параметров в переменных окружения:

- ANSIBLE_NET_USERNAME - для переменной username
- ANSIBLE_NET_PASSWORD - password
- ANSIBLE_NET_SSH_KEYFILE - ssh_keyfile
- ANSIBLE_NET_AUTHORIZE - authorize
- ANSIBLE_NET_AUTH_PASS - auth_pass

АРГУМЕНТ PROVIDER

Приоритетность значений в порядке возрастания приоритетности:

- значения по умолчанию
- значения переменных окружения
- параметр provider
- аргументы задачи (task)

ПОДГОТОВКА К РАБОТЕ С СЕТЕВЫМИ МОДУЛЯМИ

В следующих разделах рассматривается работа с модулями ios_command, ios_facts и ios_config. Для того, чтобы все примеры playbook работали, надо создать несколько файлов (проверить, что они есть).

ПОДГОТОВКА К РАБОТЕ С СЕТЕВЫМИ МОДУЛЯМИ

Инвентарный файл myhosts:

```
[cisco-routers]
192.168.100.1
192.168.100.2
192.168.100.3

[cisco-switches]
192.168.100.100
```

ПОДГОТОВКА К РАБОТЕ С СЕТЕВЫМИ МОДУЛЯМИ

Конфигурационный файл ansible.cfg:

```
[defaults]

inventory = ./myhosts

remote_user = cisco
ask_pass = True
```

ПОДГОТОВКА К РАБОТЕ С СЕТЕВЫМИ МОДУЛЯМИ

В файле group_vars/all.yml надо создать переменную cli, чтобы не указывать каждый раз все параметры, которые нужно передать аргументу provider:

```
cli:
  host: "{{ inventory_hostname }}"
  username: "cisco"
  password: "cisco"
  transport: cli
  authorize: yes
  auth_pass: "cisco"
```

МОДУЛЬ IOS_COMMAND

МОДУЛЬ IOS_COMMAND

Модуль **ios_command** - отправляет команду show на устройство под управлением IOS и возвращает результат выполнения команды.

Модуль ios_command не поддерживает отправку команд в конфигурационном режиме. Для этого используется отдельный модуль - ios_config.

МОДУЛЬ IOS_COMMAND

Модуль ios_command поддерживает такие параметры:

- commands - список команд, которые надо отправить на устройство
- wait_for (или waitfor) - список условий на которые надо проверить вывод команды. Задача ожидает выполнения всех условий. Если после указанного количества попыток выполнения команды условия не выполняются, будет считаться, что задача выполнена неудачно.

МОДУЛЬ IOS_COMMAND

Модуль ios_command поддерживает такие параметры:

- match - этот параметр используется вместе с wait_for для указания политики совпадения. Если параметр match установлен в all, должны выполниться все условия в wait_for. Если параметр равен any, достаточно чтобы выполнилось одно из условий.
- retries - указывает количество попыток выполнить команду, прежде чем она будет считаться невыполненной. По умолчанию - 10 попыток.
- interval - интервал в секундах между повторными попытками выполнить команду. По умолчанию - 1 секунда.

МОДУЛЬ IOS_COMMAND

Перед отправкой самой команды, модуль:

- выполняет аутентификацию по SSH,
- переходит в режим enable
- выполняет команду `terminal length 0`, чтобы вывод команд show отражался полностью, а не постранично.
- выполняет команду `terminal width 512`

МОДУЛЬ IOS_COMMAND

Пример использования модуля ios_command (playbook 1_ios_command.yml):

```
- name: Run show commands on routers
  hosts: cisco-routers
  gather_facts: false
  connection: local

  tasks:

    - name: run sh ip int br
      ios_command:
        commands: show ip int br
        provider: "{{ cli }}"
      register: sh_ip_int_br_result

    - name: Debug registered var
      debug: var=sh_ip_int_br_result.stdout_lines
```

МОДУЛЬ IOS_COMMAND

Модуль ios_command ожидает параметры:

- commands - список команд, которые нужно отправить на устройство
- provider - словарь с параметрами подключения
  - в нашем случае, он указан в файле group_vars/all.yml

Обратите внимание, что параметр register находится на одном уровне с именем задачи и модулем, а не на уровне параметров модуля ios_command.

Результат выполнения playbook:

```
$ ansible-playbook 1_ios_command.yml
```

```
SSH password:

PLAY [Run show commands on routers] ********************************************

TASK [run sh ip int br] ********************************************************
ok: [192.168.100.1]
ok: [192.168.100.2]
ok: [192.168.100.3]

TASK [Debug registered var] ****************************************************
ok: [192.168.100.1] => {
    "sh_ip_int_br_result.stdout_lines": [
        [
            "Interface                  IP-Address      OK? Method Status                Protocol",
            "Ethernet0/0                192.168.100.1   YES NVRAM  up                    up      ",
            "Ethernet0/1                192.168.200.1   YES NVRAM  up                    up      ",
            "Ethernet0/2                unassigned      YES manual administratively down down    ",
            "Ethernet0/3                unassigned      YES manual up                    up      ",
            "Loopback0                  10.1.1.1        YES manual up                    up      "
        ]
    ]
}
ok: [192.168.100.2] => {
    "sh_ip_int_br_result.stdout_lines": [
        [
            "Interface                  IP-Address      OK? Method Status                Protocol",
            "Ethernet0/0                192.168.100.2   YES manual up                    up      ",
            "Ethernet0/1                unassigned      YES unset  administratively down down    ",
            "Ethernet0/2                192.168.200.1   YES manual administratively down down    ",
            "Ethernet0/3                unassigned      YES manual up                    up      ",
            "Loopback0                  10.1.1.1        YES manual up                    up      "
        ]
    ]
}
ok: [192.168.100.3] => {
    "sh_ip_int_br_result.stdout_lines": [
        [
            "Interface                  IP-Address      OK? Method Status                Protocol",
            "Ethernet0/0                192.168.100.3   YES manual up                    up      ",
            "Ethernet0/1                unassigned      YES unset  administratively down down    ",
            "Ethernet0/2                192.168.200.1   YES manual administratively down down    ",
            "Ethernet0/3                unassigned      YES manual up                    up      ",
            "Loopback0                  10.1.1.1        YES manual up                    up      ",
            "Loopback10                 10.255.3.3      YES manual up                    up      "
        ]
    ]
}

PLAY RECAP *********************************************************************
192.168.100.1              : ok=2    changed=0    unreachable=0    failed=0
192.168.100.2              : ok=2    changed=0    unreachable=0    failed=0
192.168.100.3              : ok=2    changed=0    unreachable=0    failed=0
```

ВЫПОЛНЕНИЕ НЕСКОЛЬКИХ КОМАНД

Playbook 2_ios_command.yml выполняет несколько команд и получает их вывод:

```
- name: Run show commands on routers
  hosts: cisco-routers
  gather_facts: false
  connection: local

  tasks:

    - name: run show commands
      ios_command:
        commands:
          - show ip int br
          - sh ip route
        provider: "{{ cli }}"
      register: show_result

    - name: Debug registered var
      debug: var=show_result.stdout_lines
```

ВЫПОЛНЕНИЕ НЕСКОЛЬКИХ КОМАНД

Результат выполнения playbook (вывод сокращен):

```
$ ansible-playbook 2_ios_command.yml
```

```
SSH password:

PLAY [Run show commands on routers] ******************************************

TASK [run show commands] *****************************************************
ok: [192.168.100.3]
ok: [192.168.100.1]
ok: [192.168.100.2]

TASK [Debug registered var] **************************************************
ok: [192.168.100.1] => {
    "show_result.stdout_lines": [
        [
            "Interface                  IP-Address      OK? Method Status                Protocol",
            "Ethernet0/0                192.168.100.1   YES NVRAM  up                    up      ",
            "Ethernet0/1                192.168.200.1   YES NVRAM  up                    up      ",
            "Ethernet0/2                unassigned      YES manual administratively down down    ",
            "Ethernet0/3                unassigned      YES manual up                    up      ",
            "Loopback0                  10.1.1.1        YES manual up                    up      "
        ],
        [
            "Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP",
            "       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area ",
            "       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2",
            "       E1 - OSPF external type 1, E2 - OSPF external type 2",
            "       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2",
            "       ia - IS-IS inter area, * - candidate default, U - per-user static route",
            "       o - ODR, P - periodic downloaded static route, H - NHRP, l - LISP",
            "       + - replicated route, % - next hop override",
            "",
            "Gateway of last resort is not set",
            "",
            "      10.0.0.0/32 is subnetted, 2 subnets",
            "C        10.1.1.1 is directly connected, Loopback0",
            "D        10.255.3.3 [90/409600] via 192.168.100.3, 02:04:51, Ethernet0/0",
            "      192.168.100.0/24 is variably subnetted, 2 subnets, 2 masks",
            "C        192.168.100.0/24 is directly connected, Ethernet0/0",
            "L        192.168.100.1/32 is directly connected, Ethernet0/0",
            "      192.168.200.0/24 is variably subnetted, 2 subnets, 2 masks",
            "C        192.168.200.0/24 is directly connected, Ethernet0/1",
            "L        192.168.200.1/32 is directly connected, Ethernet0/1"
        ]
    ]
}
```

ВЫПОЛНЕНИЕ НЕСКОЛЬКИХ КОМАНД

Если модулю передаются несколько команд, результат выполнения команд находится в переменных stdout и stdout_lines в списке. Вывод будет в том порядке, в котором команды описаны в задаче.

Засчет этого, например, можно вывести результат выполнения первой команды, указав:

```
- name: Debug registered var
  debug: var=show_result.stdout_lines[0]
```

ОБРАБОТКА ОШИБОК

В модуле встроено распознание ошибок. Поэтому, если команда выполнена с ошибкой, модуль отобразит, что возникла ошибка.

Например, если сделать ошибку в команде, и запустить playbook еще раз

```
$ ansible-playbook 2_ios_command.yml
```

ОБРАБОТКА ОШИБОК

```
SSH password:

PLAY [Run show commands on routers] ********************************************

TASK [run sh ip int br] ********************************************************
fatal: [192.168.100.1]: FAILED! => {"changed": false, "failed": true, "msg": "matched
error in response: shw ip int br\r\n     ^\r\n% Invalid input detected at '^' marker.\
r\n\r\nR1#"}
fatal: [192.168.100.2]: FAILED! => {"changed": false, "failed": true, "msg": "matched
error in response: shw ip int br\r\n     ^\r\n% Invalid input detected at '^' marker.\
r\n\r\nR2#"}
fatal: [192.168.100.3]: FAILED! => {"changed": false, "failed": true, "msg": "matched
error in response: shw ip int br\r\n     ^\r\n% Invalid input detected at '^' marker.\
r\n\r\nR3#"}
        to retry, use: --limit @/home/nata/pyneng_course/chapter15/2_ios_command.retry

PLAY RECAP *********************************************************************
192.168.100.1              : ok=0    changed=0    unreachable=0    failed=1
192.168.100.2              : ok=0    changed=0    unreachable=0    failed=1
192.168.100.3              : ok=0    changed=0    unreachable=0    failed=1
```

ОБРАБОТКА ОШИБОК

Ansible обнаружил ошибку и возвращает сообщение ошибки. В данном случае - 'Invalid input'.

Аналогичным образом модуль обнаруживает ошибки:

- Ambiguous command
- Incomplete command

WAIT_FOR

Пример playbook (файл 3_ios_command_wait_for.yml):

```
- name: Run show commands on routers
  hosts: cisco-routers
  gather_facts: false
  connection: local

  tasks:

    - name: run show commands
      ios_command:
        commands: ping 192.168.100.100
        wait_for:
          - result[0] contains 'Success rate is 100 percent'
        provider: "{{ cli }}"
```

WAIT_FOR

В playbook всего одна задача, которая отправляет команду ping 192.168.100.100 и проверяет есть ли в выводе команды фраза 'Success rate is 100 percent'.

Если в выводе команды содержится эта фраза, задача считается корректно выполненой.

Запуск playbook:

```
$ ansible-playbook 3_ios_command_wait_for.yml -v
```

WAIT_FOR

```
Using /home/vagrant/repos/pyneng-online-jun-jul-2017/examples/15_ansible/3_network_module
s/ios_command/ansible.cfg as config file

PLAY [Run show commands on routers] ********************************************************

TASK [run show commands] *******************************************************************
ok: [192.168.100.1] => {"changed": false, "failed": false, "stdout": ["Type escape sequen
ce to abort.\nSending 5, 100-byte ICMP Echos to 192.168.100.100, timeout is 2 seconds:\n!
!!!!\nSuccess rate is 100 percent (5/5), round-trip min/avg/max = 1/1/1 ms"], "stdout_lin
es": [["Type escape sequence to abort.", "Sending 5, 100-byte ICMP Echos to 192.168.100.1
00, timeout is 2 seconds:", "!!!!!", "Success rate is 100 percent (5/5), round-trip min/a
vg/max = 1/1/1 ms"]]}
ok: [192.168.100.2] => {"changed": false, "failed": false, "stdout": ["Type escape sequen
ce to abort.\nSending 5, 100-byte ICMP Echos to 192.168.100.100, timeout is 2 seconds:\n!
!!!!\nSuccess rate is 100 percent (5/5), round-trip min/avg/max = 1/1/1 ms"], "stdout_lin
es": [["Type escape sequence to abort.", "Sending 5, 100-byte ICMP Echos to 192.168.100.1
00, timeout is 2 seconds:", "!!!!!", "Success rate is 100 percent (5/5), round-trip min/a
vg/max = 1/1/1 ms"]]}
ok: [192.168.100.3] => {"changed": false, "failed": false, "stdout": ["Type escape sequen
ce to abort.\nSending 5, 100-byte ICMP Echos to 192.168.100.100, timeout is 2 seconds:\n!
!!!!\nSuccess rate is 100 percent (5/5), round-trip min/avg/max = 1/1/1 ms"], "stdout_lin
es": [["Type escape sequence to abort.", "Sending 5, 100-byte ICMP Echos to 192.168.100.1
00, timeout is 2 seconds:", "!!!!!", "Success rate is 100 percent (5/5), round-trip min/a
vg/max = 1/1/1 ms"]]}

PLAY RECAP *********************************************************************************
192.168.100.1              : ok=1    changed=0    unreachable=0    failed=0
192.168.100.2              : ok=1    changed=0    unreachable=0    failed=0
192.168.100.3              : ok=1    changed=0    unreachable=0    failed=0
```

МОДУЛЬ IOS_FACTS

МОДУЛЬ IOS_FACTS

Модуль ios_facts - собирает информацию с устройств под управлением IOS.

Информация берется из таких команд:

- dir
- show version
- show memory statistics
- show interfaces
- show ipv6 interface
- show lldp
- show lldp neighbors detail
- show running-config

МОДУЛЬ IOS_FACTS

В модуле можно указывать какие параметры собирать - можно собирать всю информацию, а можно только подмножество. По умолчанию, модуль собирает всю информацию, кроме конфигурационного файла.

Какую информацию собирать, указывается в параметре **gather_subset**.

МОДУЛЬ IOS_FACTS

Поддерживаются такие варианты (указаны также команды, которые будут выполняться на устройстве):

- **all**
- **hardware**
  - dir
  - show version
  - show memory statistics
- **config**
  - show version
  - show running-config

МОДУЛЬ IOS_FACTS

- **interfaces**
  - dir
  - show version
  - show interfaces
  - show ipv6 interface
  - show lldp
  - show lldp neighbors detail

МОДУЛЬ IOS_FACTS

Собрать все факты:

```
- ios_facts:
    gather_subset: all
    provider: "{{ cli }}"
```

МОДУЛЬ IOS_FACTS

Собрать только подмножество interfaces:

```
- ios_facts:
    gather_subset:
      - interfaces
    provider: "{{ cli }}"
```

МОДУЛЬ IOS_FACTS

Собрать всё, кроме hardware:

```
- ios_facts:
    gather_subset:
      - "!hardware"
    provider: "{{ cli }}"
```

МОДУЛЬ IOS_FACTS

Ansible собирает такие факты:

- ansible_net_all_ipv4_addresses - список IPv4 адресов на устройстве
- ansible_net_all_ipv6_addresses - список IPv6 адресов на устройстве
- ansible_net_config - конфигурация (для Cisco sh run)
- ansible_net_filesystems - файловая система устройства
- ansible_net_gather_subset - какая информация собирается (hardware, default, interfaces, config)
- ansible_net_hostname - имя устройства

МОДУЛЬ IOS_FACTS

- ansible_net_image - имя и путь ОС
- ansible_net_interfaces - словарь со всеми интерфейсами устройства. Имена интерфейсов - ключи, а данные - параметры каждого интерфейса
- ansible_net_memfree_mb - сколько свободной памяти на устройстве
- ansible_net_memtotal_mb - сколько памяти на устройстве
- ansible_net_model - модель устройства
- ansible_net_serialnum - серийный номер
- ansible_net_version - версия IOS

ИСПОЛЬЗОВАНИЕ МОДУЛЯ

Пример playbook 1_ios_facts.yml с использованием модуля ios_facts (собираются все факты):

```
- name: Collect IOS facts
  hosts: cisco-routers
  gather_facts: false
  connection: local

  tasks:

    - name: Facts
      ios_facts:
        gather_subset: all
        provider: "{{ cli }}"
```

ИСПОЛЬЗОВАНИЕ МОДУЛЯ

```
$ ansible-playbook 1_ios_facts.yml
```

```
SSH password:

PLAY [Collect IOS facts] ********************************************************

TASK [Facts] ********************************************************************
ok: [192.168.100.1]
ok: [192.168.100.2]
ok: [192.168.100.3]

PLAY RECAP **********************************************************************
192.168.100.1              : ok=1    changed=0    unreachable=0    failed=0
192.168.100.2              : ok=1    changed=0    unreachable=0    failed=0
192.168.100.3              : ok=1    changed=0    unreachable=0    failed=0
```

ИСПОЛЬЗОВАНИЕ МОДУЛЯ

Для того, чтобы посмотреть, какие именно факты собираются с устройства, можно добавить флаг -v (информация сокращена):

```
$ ansible-playbook 1_ios_facts.yml -v
Using /home/nata/pyneng_course/chapter15/ansible.cfg as config file
```

```
SSH password:

PLAY [Collect IOS facts] ********************************************************

TASK [Facts] ********************************************************************
ok: [192.168.100.1] => {"ansible_facts": {"ansible_net_all_ipv4_addresses": ["192.168.
200.1", "192.168.100.1", "10.1.1.1"], "ansible_net_all_ipv6_addresses": [], "ansible_n
et_config": "Building configuration...\n\nCurrent configuration : 6716 bytes\n!\n! Las
t configuration change at 09:09:04 UTC Sun Dec 18 2016\nversion 15.2\nno service times
```

ИСПОЛЬЗОВАНИЕ МОДУЛЯ

После того, как Ansible собрал факты с устройства, все факты доступны как переменные в playbook, шаблонах и т.д.

ИСПОЛЬЗОВАНИЕ МОДУЛЯ

Например, можно отобразить содержимое факта с помощью debug (playbook 2_ios_facts_debug.yml):

```
- name: Collect IOS facts
  hosts: 192.168.100.1
  gather_facts: false
  connection: local

  tasks:

    - name: Facts
      ios_facts:
        gather_subset: all
        provider: "{{ cli }}"

    - name: Show ansible_net_all_ipv4_addresses fact
      debug: var=ansible_net_all_ipv4_addresses

    - name: Show ansible_net_interfaces fact
      debug: var=ansible_net_interfaces['Ethernet0/0']
```

ИСПОЛЬЗОВАНИЕ МОДУЛЯ

Результат выполнения playbook:

```
$ ansible-playbook 2_ios_facts_debug.yml
```

```
SSH password:

PLAY [Collect IOS facts] ********************************************************

TASK [Facts] ********************************************************************
ok: [192.168.100.1]

TASK [Show ansible_net_all_ipv4_addresses fact] *********************************
ok: [192.168.100.1] => {
    "ansible_net_all_ipv4_addresses": [
        "192.168.200.1",
        "192.168.100.1"
    ]
}

TASK [Show fact] ****************************************************************
ok: [192.168.100.1] => {
    "ansible_net_interfaces['Ethernet0/0']": {
        "bandwidth": 10000,
        "description": null,
        "duplex": null,
        "ipv4": {
            "address": "192.168.100.1",
            "masklen": 24
        },
        "lineprotocol": "up ",
        "macaddress": "aabb.cc00.6500",
        "mediatype": null,
        "mtu": 1500,
        "operstatus": "up",
        "type": "AmdP2"
    }
}

PLAY RECAP **********************************************************************
192.168.100.1              : ok=3    changed=0    unreachable=0    failed=0
```

СОХРАНЕНИЕ ФАКТОВ

В том виде, в котором информация отображается в режиме verbose, довольно сложно понять какая информация собирается об устройствах. Для того, чтобы лучше понять какая информация собирается об устройствах, в каком формате, скопируем полученную информацию в файл.

Для этого будет использоваться модуль copy.

СОХРАНЕНИЕ ФАКТОВ

Playbook 3_ios_facts.yml:

```
- name: Collect IOS facts
  hosts: cisco-routers
  gather_facts: false
  connection: local

  tasks:

    - name: Facts
      ios_facts:
        gather_subset: all
        provider: "{{ cli }}"
      register: ios_facts_result

    #- name: Create all_facts dir
    #  file:
    #    path: ./all_facts/
    #    state: directory
    #    mode: 0755

    - name: Copy facts to files
      copy:
        content: "{{ ios_facts_result | to_nice_json }}"
        dest: "all_facts/{{inventory_hostname}}_facts.json"
```

СОХРАНЕНИЕ ФАКТОВ

Модуль copy позволяет копировать файлы с управляющего хоста (на котором установлен Ansible) на удаленный хост. Но, так как в этом случае, указан параметр `connection: local`, файлы будут скопированы на локальный хост.

Чаще всего, модуль copy используется таким образом:

```
- copy:
    src: /srv/myfiles/foo.conf
    dest: /etc/foo.conf
```

СОХРАНЕНИЕ ФАКТОВ

Но, в данном случае, нет исходного файла, содержимое которого нужно скопировать. Вместо этого, есть содержимое переменной ios_facts_result, которое нужно перенести в файл all_facts/{{inventory_hostname}}_facts.json.

Для того чтобы перенести содержимое переменной в файл, в модуле copy, вместо src, используется параметр content.

СОХРАНЕНИЕ ФАКТОВ

```
content: "{{ ios_facts_result | to_nice_json }}"
```

Параметр to_nice_json - это фильтр Jinja2, который преобразует информацию переменной в формат, в котором удобней читать информацию. Переменная в формате Jinja2 должна быть заключена в двойные фигурные скобки, а также указана в двойных кавычках.

Так как в пути dest используются имена устройств, будут сгенерированы уникальные файлы для каждого устройства.

Результат выполнения playbook:

```
$ ansible-playbook 3_ios_facts.yml
```

```
SSH password:

PLAY [Collect IOS facts] ********************************************************

TASK [Facts] ********************************************************************
ok: [192.168.100.1]
ok: [192.168.100.2]
ok: [192.168.100.3]

TASK [Copy facts to files] ******************************************************
changed: [192.168.100.3]
changed: [192.168.100.1]
changed: [192.168.100.2]

PLAY RECAP **********************************************************************
192.168.100.1              : ok=2    changed=1    unreachable=0    failed=0
192.168.100.2              : ok=2    changed=1    unreachable=0    failed=0
192.168.100.3              : ok=2    changed=1    unreachable=0    failed=0
```

СОХРАНЕНИЕ ФАКТОВ

После этого, в каталоге all_facts находятся такие файлы:

```
192.168.100.1_facts.json
192.168.100.2_facts.json
192.168.100.3_facts.json
```

СОХРАНЕНИЕ ФАКТОВ

Содержимое файла all_facts/192.168.100.1_facts.json:


```
{
    "ansible_facts": {
        "ansible_net_all_ipv4_addresses": [
            "192.168.200.1",
            "192.168.100.1",
            "10.1.1.1"
        ],
        "ansible_net_all_ipv6_addresses": [],
        "ansible_net_config": "Building configuration...\n\nCurrent configuration :
...
```

СОХРАНЕНИЕ ФАКТОВ

Сохранение информации об устройствах, не только поможет разобраться, какая информация собирается, но и может быть полезным для дальнейшего использования информации. Например, можно использовать факты об устройстве в шаблоне.

При повторном выполнении playbook, Ansible не будет изменять информацию в файлах, если факты об устройстве не изменились

СОХРАНЕНИЕ ФАКТОВ

Если информация изменилась, для соответствующего устройства, будет выставлен статус changed. Таким образом, по выполнению playbook всегда понятно, когда какая-то информация изменилась.

Повторный запуск playbook (без изменений):

```
$ ansible-playbook 3_ios_facts.yml
```

```
SSH password:

PLAY [Collect IOS facts] ********************************************************

TASK [Facts] ********************************************************************
ok: [192.168.100.1]
ok: [192.168.100.3]
ok: [192.168.100.2]

TASK [Copy facts to files] ******************************************************
ok: [192.168.100.2]
ok: [192.168.100.1]
ok: [192.168.100.3]

PLAY RECAP **********************************************************************
192.168.100.1              : ok=2    changed=0    unreachable=0    failed=0
192.168.100.2              : ok=2    changed=0    unreachable=0    failed=0
192.168.100.3              : ok=2    changed=0    unreachable=0    failed=0
```

МОДУЛЬ IOS_CONFIG

МОДУЛЬ IOS_CONFIG

Модуль ios_config - позволяет настраивать устройства под управлением IOS, а также, генерировать шаблоны конфигураций или отправлять команды на основании шаблона.

МОДУЛЬ IOS_CONFIG

Параметры модуля:

- **after** - какие действия выполнить после команд
- **before** - какие действия выполнить до команд
- **backup** - параметр, который указывает нужно ли делать резервную копию текущей конфигурации устройства перед внесением изменений. Файл будет копироваться в каталог backup, относительно каталога в котором находится playbook
- **config** - параметр, который позволяет указать базовый файл конфигурации, с которым будут сравниваться изменения. Если он указан, модуль не будет скачивать конфигурацию с устройства.

МОДУЛЬ IOS_CONFIG

- **defaults** - параметр указывает нужно ли собирать всю информацию с устройства, в том числе, и значения по умолчанию. Если включить этот параметр, то модуль будет собирать текущую кофигурацию с помощью команды sh run all. По умолчанию этот параметр отключен и конфигурация проверяется командой sh run
- **lines (commands)** - список команд, которые должны быть настроены. Команды нужно указывать без сокращений и ровно в том виде, в котором они будут в конфигурации.
- **match** - параметр указывает как именно нужно сравнивать команды

МОДУЛЬ IOS_CONFIG

- **parents** - название секции, в которой нужно применить команды. Если команда находится внутри вложенной секции, нужно указывать весь путь. Если этот параметр не указан, то считается, что команда должны быть в глобальном режиме конфигурации
- **replace** - параметр указывает как выполнять настройку устройства
- **save** - сохранять ли текущую конфигурацию в стартовую. По умолчанию конфигурация не сохраняется
- **src** - параметр указывает путь к файлу, в котором находится конфигурация или шаблон конфигурации. Взаимоисключающий параметр с lines (то есть, можно указывать или lines или src). Заменяет модуль ios_template, который скоро будет удален.

МОДУЛЬ IOS_CONFIG

ПАРАМЕТР LINES (COMMANDS)

LINES (COMMANDS)

Самый простой способ использовать модуль ios_config - отправлять команды глобального конфигурационного режима с параметром lines.

Для параметра lines есть alias commands, то есть, можно вместо lines писать commands.

LINES (COMMANDS)

Пример playbook 1_ios_config_lines.yml:

```
- name: Run cfg commands on routers
  hosts: cisco-routers
  gather_facts: false
  connection: local

  tasks:

    - name: Config password encryption
      ios_config:
        lines:
          - service password-encryption
        provider: "{{ cli }}"
```

LINES (COMMANDS)

Результат выполнения playbook:

```
$ ansible-playbook 1_ios_config_lines.yml
```

```
PLAY [Run cfg commands on routers] ****************************************

TASK [Config password encryption] ****************************************
changed: [192.168.100.1]
changed: [192.168.100.3]
changed: [192.168.100.2]

PLAY RECAP ****************************************************************
192.168.100.1              : ok=1    changed=1    unreachable=0    failed=0
192.168.100.2              : ok=1    changed=1    unreachable=0    failed=0
192.168.100.3              : ok=1    changed=1    unreachable=0    failed=0
```

LINES (COMMANDS)

Ansible выполняет такие команды:

- terminal length 0
- enable
- show running-config - чтобы проверить есть ли эта команда на устройстве. Если команда есть, задача выполняться не будет. Если команды нет, задача выполнится
- если команды, которая указана в задаче нет в конфигурации:
  - configure terminal
  - service password-encryption
  - end

LINES (COMMANDS)

Так как модуль каждый раз проверяет конфигурацию, прежде чем применит команду, модуль идемпотентен. То есть, если ещё раз запустить playbook, изменения не будут выполнены:

```
$ ansible-playbook 1_ios_config_lines.yml
```

```
SSH password:

PLAY [Run cfg commands on routers] ********************************************

TASK [Config password encryption] *********************************************
ok: [192.168.100.2]
ok: [192.168.100.1]
ok: [192.168.100.3]

PLAY RECAP ********************************************************************
192.168.100.1              : ok=1    changed=0    unreachable=0    failed=0
192.168.100.2              : ok=1    changed=0    unreachable=0    failed=0
192.168.100.3              : ok=1    changed=0    unreachable=0    failed=0
```

LINES (COMMANDS)

Параметр lines позволяет отправлять и несколько команд (playbook 1_ios_config_mult_lines.yml):

```
- name: Run cfg commands on routers
  hosts: cisco-routers
  gather_facts: false
  connection: local

  tasks:

    - name: Send config commands
      ios_config:
        lines:
          - service password-encryption
          - no ip http server
          - no ip http secure-server
          - no ip domain lookup
        provider: "{{ cli }}"
```

LINES (COMMANDS)

Результат выполнения:

```
$ ansible-playbook 1_ios_config_mult_lines.yml
```

```
SSH password:

PLAY [Run cfg commands on routers] ***********************************************

TASK [Send config commands] ******************************************************
changed: [192.168.100.3]
changed: [192.168.100.1]
changed: [192.168.100.2]

PLAY RECAP ***********************************************************************
192.168.100.1              : ok=1    changed=1    unreachable=0    failed=0
192.168.100.2              : ok=1    changed=1    unreachable=0    failed=0
192.168.100.3              : ok=1    changed=1    unreachable=0    failed=0
```

МОДУЛЬ IOS_CONFIG

ПАРАМЕТР PARENTS

PARENTS

Параметр parents используется, чтобы указать в каком подрежиме применить команды.

Например, необходимо применить такие команды:

```
line vty 0 4
 login local
 transport input ssh
```

PARENTS

В таком случае, playbook 2_ios_config_parents_basic.yml будет выглядеть так:

```
- name: Run cfg commands on routers
  hosts: cisco-routers
  gather_facts: false
  connection: local

  tasks:

    - name: Config line vty
      ios_config:
        parents:
          - line vty 0 4
        lines:
          - login local
          - transport input ssh
        provider: "{{ cli }}"
```

PARENTS

Запуск будет выполняться аналогично предыдущим playbook:

```
$ ansible-playbook 2_ios_config_parents_basic.yml
```

```
SSH password:

PLAY [Run cfg commands on routers] ***************************************

TASK [Config line vty] ***************************************************
changed: [192.168.100.1]
changed: [192.168.100.2]
changed: [192.168.100.3]

PLAY RECAP ***************************************************************
192.168.100.1              : ok=1    changed=1    unreachable=0    failed=0
192.168.100.2              : ok=1    changed=1    unreachable=0    failed=0
192.168.100.3              : ok=1    changed=1    unreachable=0    failed=0
```

PARENTS

Если команда находится в нескольких вложенных режимах, подрежимы указываются в списке parents.

Например, необходимо выполнить такие команды:

```
policy-map OUT_QOS
 class class-default
  shape average 100000000 1000000
```

PARENTS

Тогда playbook 2_ios_config_parents_mult.yml будет выглядеть так:

```
- name: Run cfg commands on routers
  hosts: cisco-routers
  gather_facts: false
  connection: local

  tasks:

    - name: Config QoS policy
      ios_config:
        parents:
          - policy-map OUT_QOS
          - class class-default
        lines:
          - shape average 100000000 1000000
        provider: "{{ cli }}"
```

МОДУЛЬ IOS_CONFIG

ОТОБРАЖЕНИЕ ОБНОВЛЕНИЙ

ОТОБРАЖЕНИЕ ОБНОВЛЕНИЙ

Playbook 2_ios_config_parents_basic.yml:

```
- name: Run cfg commands on routers
  hosts: cisco-routers
  gather_facts: false
  connection: local

  tasks:

    - name: Config line vty
      ios_config:
        parents:
          - line vty 0 4
        lines:
          - login local
          - transport input ssh
        provider: "{{ cli }}"
```

ОТОБРАЖЕНИЕ ОБНОВЛЕНИЙ

Например, можно выполнить playbook с флагом verbose:

```
$ ansible-playbook 2_ios_config_parents_basic.yml -v
```

```
Using /home/nata/pyneng_course/chapter15/ansible.cfg as config file
SSH password:

PLAY [Run cfg commands on routers] *********************************************

TASK [Config line vty] *********************************************************
ok: [192.168.100.3] => {"changed": false, "warnings": []}
changed: [192.168.100.1] => {"changed": true, "updates": ["line vty 0 4", "transport
 input ssh"], "warnings": []}
ok: [192.168.100.2] => {"changed": false, "warnings": []}

PLAY RECAP *********************************************************************
192.168.100.1              : ok=1    changed=1    unreachable=0    failed=0
192.168.100.2              : ok=1    changed=0    unreachable=0    failed=0
192.168.100.3              : ok=1    changed=0    unreachable=0    failed=0
```

ОТОБРАЖЕНИЕ ОБНОВЛЕНИЙ

В выводе, в поле updates видно, какие именно команды Ansible отправил на устройство. Изменения были выполнены только на маршрутизаторе 192.168.100.1.

Обратите внимание, что команда login local не отправлялась, так как она настроена.

Поле updates в выводе есть только в том случае, когда есть изменения.

ОТОБРАЖЕНИЕ ОБНОВЛЕНИЙ

В режиме verbose, информация видна обо всех устройствах. Но, было бы удобней, чтобы информация отображалась только для тех устройств, для которых произошли изменения.

ОТОБРАЖЕНИЕ ОБНОВЛЕНИЙ

Новый playbook 3_ios_config_debug.yml на основе 2_ios_config_parents_basic.yml:

```
- name: Run cfg commands on routers
  hosts: cisco-routers
  gather_facts: false
  connection: local

  tasks:

    - name: Config line vty
      ios_config:
        parents:
          - line vty 0 4
        lines:
          - login local
          - transport input ssh
        provider: "{{ cli }}"
      register: cfg

    - name: Show config updates
      debug: var=cfg.updates
      when: cfg.changed
```

ОТОБРАЖЕНИЕ ОБНОВЛЕНИЙ

Изменения в playbook:

- результат работы первой задачи сохраняется в переменную **cfg**.
- в следующей задаче модуль **debug** выводит содержимое поля **updates**.
  - но так как поле updates в выводе есть только в том случае, когда есть изменения, ставится условие when, которое проверяет были ли изменения
  - задача будет выполняться только если на устройстве были внесены изменения.
  - вместо `when: cfg.changed` можно написать `when: cfg.changed == true`

ОТОБРАЖЕНИЕ ОБНОВЛЕНИЙ

Если запустить повторно playbook, когда изменений не было, задача Show config updates, пропускается:

```
$ ansible-playbook 3_ios_config_debug.yml
```

```
SSH password:

PLAY [Run cfg commands on routers] ***********************************************

TASK [Config line vty] ***********************************************************
ok: [192.168.100.2]
ok: [192.168.100.3]
ok: [192.168.100.1]

TASK [Show config updates] *******************************************************
skipping: [192.168.100.1]
skipping: [192.168.100.2]
skipping: [192.168.100.3]

PLAY RECAP ***********************************************************************
192.168.100.1              : ok=1    changed=0    unreachable=0    failed=0
192.168.100.2              : ok=1    changed=0    unreachable=0    failed=0
192.168.100.3              : ok=1    changed=0    unreachable=0    failed=0
```

ОТОБРАЖЕНИЕ ОБНОВЛЕНИЙ

Если внести изменения в конфигурацию маршрутизатора 192.168.100.1 (изменить transport input ssh на transport input all):

```
$ ansible-playbook 3_ios_config_debug.yml
```

```
SSH password:

PLAY [Run cfg commands on routers] ********************************************

TASK [Config line vty] ********************************************************
ok: [192.168.100.2]
changed: [192.168.100.1]
ok: [192.168.100.3]

TASK [Show config updates] ****************************************************
ok: [192.168.100.1] => {
    "cfg.updates": [
        "line vty 0 4",
        "transport input ssh"
    ]
}
skipping: [192.168.100.2]
skipping: [192.168.100.3]

PLAY RECAP ********************************************************************
192.168.100.1              : ok=2    changed=1    unreachable=0    failed=0
192.168.100.2              : ok=1    changed=0    unreachable=0    failed=0
192.168.100.3              : ok=1    changed=0    unreachable=0    failed=0
```

Теперь второе задание отображает информацию о том, какие именно изменения были внесены на маршрутизаторе.

МОДУЛЬ IOS_CONFIG

SAVE_WHEN

SAVE_WHEN

Параметр **save_when** позволяет указать нужно ли сохранять текущую конфигурацию в стартовую.

Доступные варианты значений:

- always - всегда сохранять конфигурацию (в этом случае флаг modified будет равен True)
- never (по умолчанию) - не сохранять конфигурацию
- modified - в этом случае конфигурация сохраняется только при наличии изменений

SAVE_WHEN

К сожалению, на данный момент (версия ansible 2.4), этот параметр не отрабатывает корректно, так как на устройство отправляется команда copy running-config startup-config, но, при этом, не отправляется подтверждение на сохранение. Из-за этого, при запуске playbook с параметром save_when выставленным в always или modified, появляется такая ошибка:

```
fatal: [192.168.100.2]: FAILED! => {"changed": false, "failed": true,
"msg": "timeout trying to send command: b'copy running-config startup-config'", "rc": 1}
```

SAVE_WHEN

Исправить это достаточно легко, настроив в IOS:

```
file prompt quiet
```

По умолчанию настроено `file prompt alert`

SAVE_WHEN

Playbook 4_ios_config_save_when.yml

```
- name: Run cfg commands on routers
  hosts: cisco-routers
  gather_facts: false
  connection: local

  tasks:

    - name: Config line vty
      ios_config:
        parents:
          - line vty 0 4
        lines:
          - login local
          - transport input ssh
        save_when: modified
        provider: "{{ cli }}"
```

SAVE_WHEN

Вариант самостоятельного сохранения 4_ios_config_save.yml:

```
- name: Run cfg commands on routers
  hosts: cisco-routers
  gather_facts: false
  connection: local

  tasks:

    - name: Config line vty
      ios_config:
        parents:
          - line vty 0 4
        lines:
          - login local
          - transport input ssh
        provider: "{{ cli }}"
      register: cfg

    - name: Save config
      ios_command:
        commands:
          - write
        provider: "{{ cli }}"
      when: cfg.changed
```

SAVE_WHEN

МОДУЛЬ IOS_CONFIG

ПАРАМЕТР BACKUP

BACKUP

Параметр **backup** указывает нужно ли делать резервную копию текущей конфигурации устройства перед внесением изменений. Файл будет копироваться в каталог backup, относительно каталога в котором находится playbook (если каталог не существует, он будет создан).

BACKUP

Playbook 5_ios_config_backup.yml:

```
- name: Run cfg commands on routers
  hosts: cisco-routers
  gather_facts: false
  connection: local

  tasks:

    - name: Config line vty
      ios_config:
        parents:
          - line vty 0 4
        lines:
          - login local
          - transport input ssh
        backup: yes
        provider: "{{ cli }}"
```

BACKUP

Теперь, каждый раз, когда выполняется playbook (даже если не нужно вносить изменения в конфигурацию), в каталог backup будет копироваться текущая конфигурация:

```
$ ansible-playbook 5_ios_config_backup.yml -v
```

```
Using /home/nata/pyneng_course/chapter15/ansible.cfg as config file
SSH password:

PLAY [Run cfg commands on routers] *********************************************

TASK [Config line vty] *********************************************************
ok: [192.168.100.1] => {"backup_path": "/home/nata/pyneng_course/chapter15/backup/192
.168.100.1_config.2016-12-10@12:35:38", "changed": false, "warnings": []}
ok: [192.168.100.3] => {"backup_path": "/home/nata/pyneng_course/chapter15/backup/192
.168.100.3_config.2016-12-10@12:35:38", "changed": false, "warnings": []}
ok: [192.168.100.2] => {"backup_path": "/home/nata/pyneng_course/chapter15/backup/192
.168.100.2_config.2016-12-10@12:35:38", "changed": false, "warnings": []}

PLAY RECAP *********************************************************************
192.168.100.1              : ok=1    changed=0    unreachable=0    failed=0
192.168.100.2              : ok=1    changed=0    unreachable=0    failed=0
192.168.100.3              : ok=1    changed=0    unreachable=0    failed=0
```

BACKUP

В каталоге backup теперь находятся файлы такого вида (при каждом запуске playbook они перезаписываются):

```
192.168.100.1_config.2016-12-10@10:42:34
192.168.100.2_config.2016-12-10@10:42:34
192.168.100.3_config.2016-12-10@10:42:34
```

МОДУЛЬ IOS_CONFIG

ПАРАМЕТР DEFAULTS

DEFAULTS

Параметр **defaults** указывает нужно ли собирать всю информацию с устройства, в том числе и значения по умолчанию. Если включить этот параметр, модуль будет собирать текущую кофигурацию с помощью команды sh run all. По умолчанию этот параметр отключен и конфигурация проверяется командой sh run.

Этот параметр полезен в том случае, если в настройках указывается команда, которая не видна в конфигурации. Например, такое может быть, когда указан параметр, который и так используется по умолчанию.

DEFAULTS

Если не использовать параметр defaults, и указать команду, которая настроена по умолчанию, то при каждом запуске playbook, будут вноситься изменения.

Присходит это потому, что Ansible каждый раз вначале проверяет наличие команд в соответствующем режиме. Если команд нет, то соответствующая задача выполняется.

DEFAULTS

Например, в таком playbook, каждый раз будут вноситься изменения (попробуйте запустить его самостоятельно):

```
- name: Run cfg commands on routers
  hosts: cisco-routers
  gather_facts: false
  connection: local

  tasks:

    - name: Config interface
      ios_config:
        parents:
          - interface Ethernet0/2
        lines:
          - ip address 192.168.200.1 255.255.255.0
          - ip mtu 1500
        provider: "{{ cli }}"
```

DEFAULTS

Если добавить параметр `defaults: yes`, изменения уже не будут внесены, если не хватало только команды ip mtu 1500 (playbook 6_ios_config_defaults.yml):

```
- name: Run cfg commands on routers
  hosts: cisco-routers
  gather_facts: false
  connection: local

  tasks:

    - name: Config interface
      ios_config:
        parents:
          - interface Ethernet0/2
        lines:
          - ip address 192.168.200.1 255.255.255.0
          - ip mtu 1500
        defaults: yes
        provider: "{{ cli }}"
```

DEFAULTS

Запуск playbook:

```
$ ansible-playbook 6_ios_config_defaults.yml
```

```
SSH password:

PLAY [Run cfg commands on routers] ******************************************

TASK [Config interface] *****************************************************
ok: [192.168.100.3]
ok: [192.168.100.1]
ok: [192.168.100.2]

PLAY RECAP ******************************************************************
192.168.100.1              : ok=1    changed=0    unreachable=0    failed=0
192.168.100.2              : ok=1    changed=0    unreachable=0    failed=0
192.168.100.3              : ok=1    changed=0    unreachable=0    failed=0
```

МОДУЛЬ IOS_CONFIG

ПАРАМЕТР AFTER

AFTER

Параметр **after** указывает какие команды выполнить после команд в списке lines (или commands).

Команды, которые указаны в параметре after:

- выполняются только если должны быть внесены изменения.
- при этом они будут выполнены, независимо от того есть они в конфигурации или нет.

AFTER

Параметр after очень полезен в ситуациях, когда необходимо выполнить команду, которая не сохраняется в конфигурации.

Например, команда no shutdown не сохраняется в конфигурации маршрутизатора. И, если добавить её в список lines, изменения будут вноситься каждый раз, при выполнении playbook.

Но, если написать команду no shutdown в списке after, то она будет применена только в том случае, если нужно вносить изменения (согласно списка lines).

AFTER

Пример использования параметра after в playbook 7_ios_config_after.yml:

```
- name: Run cfg commands on router
  hosts: 192.168.100.1
  gather_facts: false
  connection: local

  tasks:

    - name: Config interface
      ios_config:
        parents:
          - interface Ethernet0/3
        lines:
          - ip address 192.168.230.1 255.255.255.0
        after:
          - no shutdown
        provider: "{{ cli }}"
```

AFTER

Первый запуск playbook, с внесением изменений:

```
$ ansible-playbook 7_ios_config_after.yml -v
```

```
Using /home/nata/pyneng_course/chapter15/ansible.cfg as config file
SSH password:

PLAY [Run cfg commands on routers] ***************************************

TASK [Config interface] *************************************************
changed: [192.168.100.1] => {"changed": true, "updates": ["interface Ethernet0/3",
 "ip address 192.168.230.1 255.255.255.0", "no shutdown"], "warnings": []}

PLAY RECAP **************************************************************
192.168.100.1              : ok=1    changed=1    unreachable=0    failed=0
```

AFTER

Второй запуск playbook (изменений нет, поэтому команда no shutdown не выполняется):

```
$ ansible-playbook 7_ios_config_after.yml -v
```

```
Using /home/nata/pyneng_course/chapter15/ansible.cfg as config file
SSH password:

PLAY [Run cfg commands on routers] *********************************************

TASK [Config interface] ********************************************************
ok: [192.168.100.1] => {"changed": false, "warnings": []}

PLAY RECAP *********************************************************************
192.168.100.1              : ok=1    changed=0    unreachable=0    failed=0
```

AFTER

С помощью after можно сохранять конфигурацию устройства (playbook 7_ios_config_after_save.yml):

```
- name: Run cfg commands on routers
  hosts: cisco-routers
  gather_facts: false
  connection: local

  tasks:

    - name: Config line vty
      ios_config:
        parents:
          - line vty 0 4
        lines:
          - login local
          - transport input ssh
        after:
          - end
          - write
        provider: "{{ cli }}"
```

AFTER

Результат выполнения playbook (изменения только на маршрутизаторе 192.168.100.1):

```
$ ansible-playbook 7_ios_config_after_save.yml -v
```

```
Using /home/nata/pyneng_course/chapter15/ansible.cfg as config file
SSH password:

PLAY [Run cfg commands on routers] ********************************************

TASK [Config line vty] ********************************************************
ok: [192.168.100.2] => {"changed": false, "warnings": []}
ok: [192.168.100.3] => {"changed": false, "warnings": []}
changed: [192.168.100.1] => {"changed": true, "updates": ["line vty 0 4", "transpo
rt input ssh", "end", "write"], "warnings": []}

PLAY RECAP ********************************************************************
192.168.100.1              : ok=1    changed=1    unreachable=0    failed=0
192.168.100.2              : ok=1    changed=0    unreachable=0    failed=0
192.168.100.3              : ok=1    changed=0    unreachable=0    failed=0
```

МОДУЛЬ IOS_CONFIG

ПАРАМЕТР BEFORE

BEFORE

Параметр **before** указывает какие действия выполнить до команд в списке lines.

Команды, которые указаны в параметре before:

- выполняются только если должны быть внесены изменения.
- при этом они будут выполнены, независимо от того есть они в конфигурации или нет.

BEFORE

Параметр before полезен в ситуациях, когда какие-то действия необходимо выполнить перед выполнением команд в списке lines.

При этом, как и after, параметр before не влияет на то, какие команды сравниваются с конфигурацией. То есть, по-прежнему, сравниваются только команды в списке lines.

BEFORE

Playbook 8_ios_config_before.yml:

```
- name: Run cfg commands on router
  hosts: 192.168.100.1
  gather_facts: false
  connection: local

  tasks:

    - name: Config ACL
      ios_config:
        before:
          - no ip access-list extended IN_to_OUT
        parents:
          - ip access-list extended IN_to_OUT
        lines:
          - permit tcp 10.0.1.0 0.0.0.255 any eq www
          - permit tcp 10.0.1.0 0.0.0.255 any eq 22
          - permit icmp any any
        provider: "{{ cli }}"
```

BEFORE

В playbook 8_ios_config_before.yml ACL IN_to_OUT сначала удалятся, с помощью параметра before, а затем создается заново.

Таким образом в ACL всегда находятся только те строки, которые заданы в списке lines.

BEFORE

Запуск playbook с изменениями:

```
$ ansible-playbook 8_ios_config_before.yml -v
```

```
Using /home/nata/pyneng_course/chapter15/ansible.cfg as config file
SSH password:

PLAY [Run cfg commands on router] ***********************************************

TASK [Config ACL] ***************************************************************
changed: [192.168.100.1] => {"changed": true, "updates": ["no ip access-list exten
ded IN_to_OUT", "ip access-list extended IN_to_OUT", "permit tcp 10.0.1.0 0.0.0.25
5 any eq www", "permit tcp 10.0.1.0 0.0.0.255 any eq 22", "permit icmp any any"],
"warnings": []}

PLAY RECAP **********************************************************************
192.168.100.1              : ok=1    changed=1    unreachable=0    failed=0
```

BEFORE

Запуск playbook без изменений (команда в списке before не выполняется):

```
$ ansible-playbook 8_ios_config_before.yml -v
```

```
Using /home/nata/pyneng_course/chapter15/ansible.cfg as config file
SSH password:

PLAY [Run cfg commands on router] ***********************************************

TASK [Config ACL] **************************************************************
ok: [192.168.100.1] => {"changed": false, "warnings": []}

PLAY RECAP *********************************************************************
192.168.100.1              : ok=1    changed=0    unreachable=0    failed=0
```

МОДУЛЬ IOS_CONFIG

ПАРАМЕТР MATCH

MATCH

Параметр **match** указывает как именно нужно сравнивать команды (что считается изменением):

- **line** - команды проверяются построчно. Этот режим используется по умолчанию
- **strict** - должны совпасть не только сами команды, но их положение относительно друг друга
- **exact** - команды должны в точности сопадать с конфигурацией и не должно быть никаких лишних строк
- **none** - модуль не будет сравнивать команды с текущей конфигурацией

MATCH: LINE

Режим `match: line` используется по умолчанию.

В этом режиме, модуль проверяет только наличие строк, перечисленных в списке lines в соответствующем режиме. При этом, не проверяется порядок строк.

MATCH: LINE

На маршрутизаторе 192.168.100.1 настроен такой ACL:

```
R1#sh run | s access
ip access-list extended IN_to_OUT
 permit tcp 10.0.1.0 0.0.0.255 any eq 22
```

MATCH: LINE

Пример использования playbook 9_ios_config_match_line.yml в режиме line:

```
- name: Run cfg commands on router
  hosts: 192.168.100.1
  gather_facts: false
  connection: local

  tasks:

    - name: Config ACL
      ios_config:
        parents:
          - ip access-list extended IN_to_OUT
        lines:
          - permit tcp 10.0.1.0 0.0.0.255 any eq www
          - permit tcp 10.0.1.0 0.0.0.255 any eq 22
          - permit icmp any any
        provider: "{{ cli }}"
```

MATCH: LINE

Результат выполнения playbook:

```
$ ansible-playbook 9_ios_config_match_line.yml -v
```

```
Using /home/nata/pyneng_course/chapter15/ansible.cfg as config file
SSH password:

PLAY [Run cfg commands on router] ***********************************************

TASK [Config ACL] ***************************************************************
changed: [192.168.100.1] => {"changed": true, "updates": ["ip access-list extended
 IN_to_OUT", "permit tcp 10.0.1.0 0.0.0.255 any eq www", "permit icmp any any"], "
warnings": []}

PLAY RECAP **********************************************************************
192.168.100.1              : ok=1    changed=1    unreachable=0    failed=0
```

MATCH: LINE

Обратите внимание, что в списке updates только две из трёх строк ACL. Так как в режиме lines модуль сравнивает команды независимо друг от друга, он обнаружил, что не хватает только двух команд из трех.

MATCH: LINE

В итоге конфигурация на маршрутизаторе выглядит так:

```
R1#sh run | s access
ip access-list extended IN_to_OUT
 permit tcp 10.0.1.0 0.0.0.255 any eq 22
 permit tcp 10.0.1.0 0.0.0.255 any eq www
 permit icmp any any
```

То есть, порядок команд поменялся. И, хотя в этом случае, это не важно, иногда это может привести совсем не к тем результатам, которые ожидались.

Если повторно запустить playbook, при такой конфигурации, он не будет выполнять изменения, так как все строки были найдены.

MATCH: EXACT

Пример, в котором порядок команд важен.

ACL на маршрутизаторе:

```
R1#sh run | s access
ip access-list extended IN_to_OUT
 permit tcp 10.0.1.0 0.0.0.255 any eq 22
 permit tcp 10.0.1.0 0.0.0.255 any eq www
 deny   ip any any
```

MATCH: EXACT

Playbook 9_ios_config_match_exact.yml (будет постепенно дополняться):

```
- name: Run cfg commands on router
  hosts: 192.168.100.1
  gather_facts: false
  connection: local

  tasks:

    - name: Config ACL
      ios_config:
        parents:
          - ip access-list extended IN_to_OUT
        lines:
          - permit tcp 10.0.1.0 0.0.0.255 any eq www
          - permit tcp 10.0.1.0 0.0.0.255 any eq 22
          - permit icmp any any
          - deny   ip any any
        provider: "{{ cli }}"
```

MATCH: EXACT

Если запустить playbook, результат будет таким:

```
$ ansible-playbook 9_ios_config_match_exact.yml -v
```

```
Using /home/nata/pyneng_course/chapter15/ansible.cfg as config file
SSH password:

PLAY [Run cfg commands on router] ***********************************************

TASK [Config ACL] ***************************************************************
changed: [192.168.100.1] => {"changed": true, "updates": ["ip access-list extended
 IN_to_OUT", "permit icmp any any", "deny ip any any"], "warnings": []}

PLAY RECAP **********************************************************************
192.168.100.1              : ok=1    changed=1    unreachable=0    failed=0
```

MATCH: EXACT

Теперь ACL выглядит так:

```
R1#sh run | s access
ip access-list extended IN_to_OUT
 permit tcp 10.0.1.0 0.0.0.255 any eq 22
 permit tcp 10.0.1.0 0.0.0.255 any eq www
 deny   ip any any
 permit icmp any any
```

Конечно же, в таком случае, последнее правило никогда не сработает.

MATCH: EXACT

Можно добавить к этому playbook параметр before и сначала удалить ACL, а затем применять команды:

```
- name: Run cfg commands on router
  hosts: 192.168.100.1
  gather_facts: false
  connection: local

  tasks:

    - name: Config ACL
      ios_config:
        before:
          - no ip access-list extended IN_to_OUT
        parents:
          - ip access-list extended IN_to_OUT
        lines:
          - permit tcp 10.0.1.0 0.0.0.255 any eq www
          - permit tcp 10.0.1.0 0.0.0.255 any eq 22
          - permit icmp any any
          - deny   ip any any
        provider: "{{ cli }}"
```

Если применить playbook к последнему состоянию маршрутизатора, то изменений не будет никаких, так как все строки уже есть.

MATCH: EXACT

Попробуем начать с такого состояния ACL:

```
R1#sh run | s access
ip access-list extended IN_to_OUT
 permit tcp 10.0.1.0 0.0.0.255 any eq 22
 permit tcp 10.0.1.0 0.0.0.255 any eq www
 deny   ip any any
```

MATCH: EXACT

Результат будет таким:

```
$ ansible-playbook 9_ios_config_match_exact.yml -v
```

```
Using /home/nata/pyneng_course/chapter15/ansible.cfg as config file
SSH password:



PLAY [Run cfg commands on router] ***********************************************

TASK [Config ACL] **************************************************************
changed: [192.168.100.1] => {"changed": true, "updates": ["no ip access-list exten
ded IN_to_OUT", "ip access-list extended IN_to_OUT", "permit icmp any any"], "warn
ings": []}

PLAY RECAP *********************************************************************
192.168.100.1              : ok=1    changed=1    unreachable=0    failed=0
```

MATCH: EXACT

И, соответственно, на маршрутизаторе:

```
R1#sh run | s access
ip access-list extended IN_to_OUT
 permit icmp any any
```

MATCH: EXACT

Теперь в ACL осталась только одна строка:

- Модуль проверил каких команд не хватает в ACL (так как режим по умолчанию match: line),
- обнаружил, что не хватает команды `permit icmp any any` и добавил её

Но, так как в playbook ACL сначала удаляется, а затем применяется список команд lines, получилось, что в итоге в ACL одна строка.

MATCH: EXACT

Поможет, в такой ситуации, вариант `match: exact`:

```
- name: Run cfg commands on router
  hosts: 192.168.100.1
  gather_facts: false
  connection: local

  tasks:

    - name: Config ACL
      ios_config:
        before:
          - no ip access-list extended IN_to_OUT
        parents:
          - ip access-list extended IN_to_OUT
        lines:
          - permit tcp 10.0.1.0 0.0.0.255 any eq www
          - permit tcp 10.0.1.0 0.0.0.255 any eq 22
          - permit icmp any any
          - deny   ip any any
        match: exact
        provider: "{{ cli }}"
```

MATCH: EXACT

Применение playbook 9_ios_config_match_exact.yml к текущему состоянию маршрутизатора (в ACL одна строка):

```
$ ansible-playbook 9_ios_config_match_exact.yml -v
```

```
Using /home/nata/pyneng_course/chapter15/ansible.cfg as config file
SSH password:

PLAY [Run cfg commands on router] ***********************************************

TASK [Config ACL] ***************************************************************
changed: [192.168.100.1] => {"changed": true, "updates": ["no ip access-list exten


ded IN_to_OUT", "ip access-list extended IN_to_OUT", "permit tcp 10.0.1.0 0.0.0.25
5 any eq www", "permit tcp 10.0.1.0 0.0.0.255 any eq 22", "permit icmp any any", "
deny   ip any any"], "warnings": []}

PLAY RECAP **********************************************************************
192.168.100.1              : ok=1    changed=1    unreachable=0    failed=0
```

MATCH: EXACT

Теперь результат такой:

```
R1#sh run | s access
ip access-list extended IN_to_OUT
 permit tcp 10.0.1.0 0.0.0.255 any eq www
 permit tcp 10.0.1.0 0.0.0.255 any eq 22
 permit icmp any any
 deny   ip any any
```

То есть, теперь ACL выглядит точно так же, как и строки в списке lines и в том же порядке.

MATCH: EXACT

Закомментируем в playbook строки с удалением ACL:

```
- name: Run cfg commands on router
  hosts: 192.168.100.1
  gather_facts: false
  connection: local

  tasks:

    - name: Config ACL
      ios_config:
        #before:
        #  - no ip access-list extended IN_to_OUT
        parents:
          - ip access-list extended IN_to_OUT
        lines:
          - permit tcp 10.0.1.0 0.0.0.255 any eq www
          - permit tcp 10.0.1.0 0.0.0.255 any eq 22
          - permit icmp any any
          - deny   ip any any
        match: exact
        provider: "{{ cli }}"
```

MATCH: EXACT

В начало ACL добавлена строка:

```
ip access-list extended IN_to_OUT
 permit udp any any
 permit tcp 10.0.1.0 0.0.0.255 any eq www
 permit tcp 10.0.1.0 0.0.0.255 any eq 22
 permit icmp any any
 deny   ip any any
```

MATCH: EXACT

То есть, последние 4 строки выглядят так, как нужно, и в том порядке, котором нужно. Но, при этом, есть лишняя строка. Для варианта match: exact - это уже несовпадение.

MATCH: EXACT

В таком варианте, playbook будет выполняться каждый раз и пытаться применить все команды из списка lines, что не будет влиять на содержимое ACL:

```
$ ansible-playbook 9_ios_config_match_exact.yml -v
```

```
Using /home/nata/pyneng_course/chapter15/ansible.cfg as config file
SSH password:

PLAY [Run cfg commands on router] ******************************************

TASK [Config ACL] **********************************************************
changed: [192.168.100.1] => {"changed": true, "updates": ["ip access-list extended
 IN_to_OUT", "permit tcp 10.0.1.0 0.0.0.255 any eq www", "permit tcp 10.0.1.0 0.0.
0.255 any eq 22", "permit icmp any any", "deny   ip any any"], "warnings": []}

PLAY RECAP *****************************************************************
192.168.100.1              : ok=1    changed=1    unreachable=0    failed=0
```

MATCH: EXACT

Это значит, что при использовании `match:exact`, важно, чтобы был какой-то способ удалить конфигурацию, если она не соответствует тому, что должно быть (или чтобы команды перезаписывались). Иначе, эта задача будет выполняться каждый раз, при запуске playbook.

MATCH: STRICT

Вариант `match: strict` не требует, чтобы объект был в точности как указано в задаче, но, команды, которые указаны в списке lines, должны быть в том же порядке.

Если указан список parents, команды в списке lines должны идти сразу за командами parents.

MATCH: STRICT

На маршрутиазаторе такой ACL:

```
ip access-list extended IN_to_OUT
 permit tcp 10.0.1.0 0.0.0.255 any eq www
 permit tcp 10.0.1.0 0.0.0.255 any eq 22
 permit icmp any any
 deny   ip any any
```

MATCH: STRICT

Playbook 9_ios_config_match_strict.yml:

```
- name: Run cfg commands on router
  hosts: 192.168.100.1
  gather_facts: false
  connection: local

  tasks:

    - name: Config ACL
      ios_config:
        before:
          - no ip access-list extended IN_to_OUT
        parents:
          - ip access-list extended IN_to_OUT
        lines:
          - permit tcp 10.0.1.0 0.0.0.255 any eq www
          - permit tcp 10.0.1.0 0.0.0.255 any eq 22
          - permit icmp any any
        match: strict
        provider: "{{ cli }}"
```

MATCH: STRICT

Выполнение playbook:

```
$ ansible-playbook 9_ios_config_match_strict.yml -v
```

```
Using /home/nata/pyneng_course/chapter15/ansible.cfg as config file
SSH password:

PLAY [Run cfg commands on router] ***********************************************

TASK [Config ACL] ***************************************************************
ok: [192.168.100.1] => {"changed": false, "warnings": []}

PLAY RECAP **********************************************************************
192.168.100.1              : ok=1    changed=0    unreachable=0    failed=0
```

MATCH: STRICT

Так как изменений не было, ACL остался таким же.

В такой же ситуации, при использовании `match: exact`, было бы обнаружено изменение и ACL бы состоял только из строк в списке lines.

MATCH: NONE

Использование `match: none` отключает идемпотентность задачи: каждый раз при выполнении playbook, будут отправляться команды, которые указаны в задаче.

MATCH: NONE

Пример playbook 9_ios_config_match_none.yml:

```
- name: Run cfg commands on router
  hosts: 192.168.100.1
  gather_facts: false
  connection: local

  tasks:

    - name: Config ACL
      ios_config:
        before:
          - no ip access-list extended IN_to_OUT
        parents:
          - ip access-list extended IN_to_OUT
        lines:
          - permit tcp 10.0.1.0 0.0.0.255 any eq www
          - permit tcp 10.0.1.0 0.0.0.255 any eq 22
          - permit icmp any any
        match: none
        provider: "{{ cli }}"
```

MATCH: NONE

Каждый раз при запуске playbook результат будет таким:

```
$ ansible-playbook 9_ios_config_match_none.yml -v
```

```
Using /home/nata/pyneng_course/chapter15/ansible.cfg as config file
SSH password:

PLAY [Run cfg commands on router] ***********************************************

TASK [Config ACL] **************************************************************
changed: [192.168.100.1] => {"changed": true, "updates": ["no ip access-list exten
ded IN_to_OUT", "ip access-list extended IN_to_OUT", "permit tcp 10.0.1.0 0.0.0.25
5 any eq www", "permit tcp 10.0.1.0 0.0.0.255 any eq 22", "permit icmp any any"],
"warnings": []}

PLAY RECAP *********************************************************************
192.168.100.1              : ok=1    changed=1    unreachable=0    failed=0
```

Использование `match: none` подходит в тех случаях, когда, независимо от текущей конфигурации, нужно отправить все команды.

МОДУЛЬ IOS_CONFIG

ПАРАМЕТР REPLACE

REPLACE

Параметр replace указывает как именно нужно заменять конфигурацию:

- **line** - в этом режиме отправляются только те команды, которых нет в конфигурации. Этот режим используется по умолчанию
- **block** - в этом режиме отправляются все команды, если хотя бы одной команды нет

REPLACE: LINE

Режим `replace: line` - это режим работы по умолчанию. В этом режиме, если были обнаружены изменения, отправляются только недостающие строки.

Например, на маршрутизаторе такой ACL:

```
R1#sh run | s access
ip access-list extended IN_to_OUT
 permit tcp 10.0.1.0 0.0.0.255 any eq www
 permit tcp 10.0.1.0 0.0.0.255 any eq 22
 permit icmp any any
```

REPLACE: LINE

Попробуем запустить такой playbook 10_ios_config_replace_line.yml:

```
- name: Run cfg commands on router
  hosts: 192.168.100.1
  gather_facts: false
  connection: local

  tasks:

    - name: Config ACL
      ios_config:
        before:
          - no ip access-list extended IN_to_OUT
        parents:
          - ip access-list extended IN_to_OUT
        lines:
          - permit tcp 10.0.1.0 0.0.0.255 any eq www
          - permit tcp 10.0.1.0 0.0.0.255 any eq 22
          - permit icmp any any
          - deny   ip any any
        provider: "{{ cli }}"
```

REPLACE: LINE

Выполнение playbook:

```
$ ansible-playbook 10_ios_config_replace_line.yml -v
```

```
Using /home/nata/pyneng_course/chapter15/ansible.cfg as config file
SSH password:

PLAY [Run cfg commands on router] ***********************************************

TASK [Config ACL] ***************************************************************
changed: [192.168.100.1] => {"changed": true, "updates": ["no ip access-list exten
ded IN_to_OUT", "ip access-list extended IN_to_OUT", "deny   ip any any"], "warnin
gs": []}

PLAY RECAP **********************************************************************
192.168.100.1              : ok=1    changed=1    unreachable=0    failed=0
```

REPLACE: LINE

После этого на маршрутизаторе такой ACL:

```
R1#sh run | s access
ip access-list extended IN_to_OUT
 deny   ip any any
```

REPLACE: LINE

В данном случае, модуль проверил каких команд не хватает в ACL (так как режим по умолчанию match: line), обнаружил, что не хватает команды `deny ip any any` и добавил её. Но, так как ACL сначала удаляется, а затем применяется список команд lines, получилось, что у теперь ACL с одной строкой.

В таких ситуациях подходит режим `replace: block`.

`REPLACE: BLOCK`

В режиме `replace: block` отправляются все команды из списка lines (и parents), если на устройстве нет хотя бы одной из этих команд.

Повторим предыдущий пример.

REPLACE: BLOCK

ACL на маршрутизаторе:

```
R1#sh run | s access
ip access-list extended IN_to_OUT
 permit tcp 10.0.1.0 0.0.0.255 any eq www
 permit tcp 10.0.1.0 0.0.0.255 any eq 22
 permit icmp any any
```

REPLACE: BLOCK

Playbook 10_ios_config_replace_block.yml:

```
- name: Run cfg commands on router
  hosts: 192.168.100.1
  gather_facts: false
  connection: local

  tasks:

    - name: Config ACL
      ios_config:
        before:
          - no ip access-list extended IN_to_OUT
        parents:
          - ip access-list extended IN_to_OUT
        lines:
          - permit tcp 10.0.1.0 0.0.0.255 any eq www
          - permit tcp 10.0.1.0 0.0.0.255 any eq 22
          - permit icmp any any
          - deny   ip any any
        replace: block
        provider: "{{ cli }}"
```

REPLACE: BLOCK

Выполнение playbook:

```
$ ansible-playbook 10_ios_config_replace_block.yml -v
```

```
Using /home/nata/pyneng_course/chapter15/ansible.cfg as config file
SSH password:

PLAY [Run cfg commands on router] ***********************************************

TASK [Config ACL] **************************************************************
changed: [192.168.100.1] => {"changed": true, "updates": ["no ip access-list exten
ded IN_to_OUT", "ip access-list extended IN_to_OUT", "permit tcp 10.0.1.0 0.0.0.25
5 any eq www", "permit tcp 10.0.1.0 0.0.0.255 any eq 22", "permit icmp any any", "
deny   ip any any"], "warnings": []}

PLAY RECAP *********************************************************************
192.168.100.1              : ok=1    changed=1    unreachable=0    failed=0
```

REPLACE: BLOCK

В результате на маршрутизаторе такой ACL:

```
R1#sh run | s access
ip access-list extended IN_to_OUT
 permit tcp 10.0.1.0 0.0.0.255 any eq www
 permit tcp 10.0.1.0 0.0.0.255 any eq 22
 permit icmp any any
 deny   ip any any
```

МОДУЛЬ IOS_CONFIG

ПАРАМЕТР SRC

SRC

Параметр **src** позволяет указывать путь к файлу конфигурации или шаблону конфигурации, которую нужно загрузить на устройство.

Этот параметр взаимоисключающий с lines (то есть, можно указывать или lines или src). Он заменяет модуль ios_template, который скоро будет удален.

КОНФИГУРАЦИЯ

Пример playbook 11_ios_config_src.yml:

```
- name: Run cfg commands on router
  hosts: 192.168.100.1
  gather_facts: false
  connection: local

  tasks:

    - name: Config ACL
      ios_config:
        src: templates/acl_cfg.txt
        provider: "{{ cli }}"
```

SRC

В файле templates/acl_cfg.txt находится такая конфигурация:

```
ip access-list extended IN_to_OUT
 permit tcp 10.0.1.0 0.0.0.255 any eq www
 permit tcp 10.0.1.0 0.0.0.255 any eq 22
 permit icmp any any
 deny   ip any any
```

SRC

Удаляем на маршрутизаторе этот ACL, если он остался с прошлых разделов, и запускаем playbook:

```
$ ansible-playbook 11_ios_config_src.yml -v
```

```
Using /home/nata/pyneng_course/chapter15/ansible.cfg as config file
SSH password:

PLAY [Run cfg commands on router] ***********************************************

TASK [Config ACL] ***************************************************************
changed: [192.168.100.1] => {"changed": true, "warnings": []}

PLAY RECAP **********************************************************************
192.168.100.1              : ok=1    changed=1    unreachable=0    failed=0
```

SRC

Неприятная особенность параметра src в том, что не видно какие изменения были внесены. Но, возможно, в следующих версиях Ansible это будет исправлено.

SRC

Теперь на маршрутизаторе настроен ACL:

```
R1#sh run | s access
ip access-list extended IN_to_OUT
 permit tcp 10.0.1.0 0.0.0.255 any eq www
 permit tcp 10.0.1.0 0.0.0.255 any eq 22
 permit icmp any any
 deny   ip any any
```

SRC

Если запустить playbook ещё раз, но никаких изменений не будет, так как этот параметр также идемпотентен:

```
$ ansible-playbook 11_ios_config_src.yml -v
```

```
Using /home/nata/pyneng_course/chapter15/ansible.cfg as config file
SSH password:

PLAY [Run cfg commands on router] ***********************************************

TASK [Config ACL] **************************************************************
ok: [192.168.100.1] => {"changed": false, "warnings": []}

PLAY RECAP *********************************************************************
192.168.100.1              : ok=1    changed=0    unreachable=0    failed=0
```

ШАБЛОН JINJA2

В параметре src можно указывать шаблон Jinja2.

Пример шаблона (файл templates/ospf.j2):

```
router ospf 1
 router-id {{ mgmnt_ip }}
 ispf
 auto-cost reference-bandwidth 10000
{% for ip in ospf_ints %}
 network {{ ip }} 0.0.0.0 area 0
{% endfor %}
```

ШАБЛОН JINJA2

В шаблоне используются две переменные:

- mgmnt_ip - IP-адрес, который будет использоваться как router-id
- ospf_ints - список IP-адресов интерфейсов, на которых нужно включить OSPF

Для настройки OSPF на трёх маршрутизаторах, нужно иметь возможность использовать разные значения этих переменных для разных устройств. Для таких задач используются файлы с переменными в каталоге host_vars.

В каталоге host_vars нужно создать такие файлы (если они ещё не созданы):

SRC

Файл host_vars/192.168.100.1:

```
hostname: london_r1
mgmnt_loopback: 100
mgmnt_ip: 10.0.0.1
ospf_ints:
  - 192.168.100.1
  - 10.0.0.1
  - 10.255.1.1
```

SRC

Файл host_vars/192.168.100.2:

```
hostname: london_r2
mgmnt_loopback: 100
mgmnt_ip: 10.0.0.2
ospf_ints:
  - 192.168.100.2
  - 10.0.0.2
  - 10.255.2.2
```

SRC

Файл host_vars/192.168.100.3:

```
hostname: london_r3
mgmnt_loopback: 100
mgmnt_ip: 10.0.0.3
ospf_ints:
  - 192.168.100.3
  - 10.0.0.3
  - 10.255.3.3
```

SRC

Теперь можно создавать playbook 11_ios_config_src_jinja.yml:

```
- name: Run cfg commands on router
  hosts: cisco-routers
  gather_facts: false
  connection: local

  tasks:

    - name: Config OSPF
      ios_config:
        src: templates/ospf.j2
        provider: "{{ cli }}"
```

SRC

Так как Ansible сам найдет переменные в каталоге host_vars, их не нужно указывать. Можно сразу запускать playbook:

```
$ ansible-playbook 11_ios_config_src_jinja.yml -v
```

```
Using /home/nata/pyneng_course/chapter15/ansible.cfg as config file
SSH password:

PLAY [Run cfg commands on router] ***********************************************

TASK [Config OSPF] *************************************************************
changed: [192.168.100.2] => {"changed": true, "warnings": []}
changed: [192.168.100.3] => {"changed": true, "warnings": []}
changed: [192.168.100.1] => {"changed": true, "warnings": []}

PLAY RECAP *********************************************************************
192.168.100.1              : ok=1    changed=1    unreachable=0    failed=0
192.168.100.2              : ok=1    changed=1    unreachable=0    failed=0
192.168.100.3              : ok=1    changed=1    unreachable=0    failed=0
```

SRC

Теперь на всех маршрутизаторах настроен OSPF:

```
R1#sh run | s ospf
router ospf 1
 router-id 10.0.0.1
 ispf
 auto-cost reference-bandwidth 10000
 network 10.0.0.1 0.0.0.0 area 0
 network 10.255.1.1 0.0.0.0 area 0
 network 192.168.100.1 0.0.0.0 area 0

R2#sh run | s ospf
router ospf 1
 router-id 10.0.0.2
 ispf
 auto-cost reference-bandwidth 10000
 network 10.0.0.2 0.0.0.0 area 0
 network 10.255.2.2 0.0.0.0 area 0
 network 192.168.100.2 0.0.0.0 area 0

router ospf 1
 router-id 10.0.0.3
 ispf
 auto-cost reference-bandwidth 10000
 network 10.0.0.3 0.0.0.0 area 0
 network 10.255.3.3 0.0.0.0 area 0
 network 192.168.100.3 0.0.0.0 area 0
```

SRC

Если запустить playbook ещё раз, но никаких изменений не будет:

```
$ ansible-playbook 11_ios_config_src_jinja.yml -v
```

```
Using /home/nata/pyneng_course/chapter15/ansible.cfg as config file
SSH password:

PLAY [Run cfg commands on router] ***********************************************

TASK [Config OSPF] *************************************************************
ok: [192.168.100.3] => {"changed": false, "warnings": []}
ok: [192.168.100.2] => {"changed": false, "warnings": []}
ok: [192.168.100.1] => {"changed": false, "warnings": []}

PLAY RECAP *********************************************************************
192.168.100.1              : ok=1    changed=0    unreachable=0    failed=0
192.168.100.2              : ok=1    changed=0    unreachable=0    failed=0
192.168.100.3              : ok=1    changed=0    unreachable=0    failed=0
```

СОВМЕЩЕНИЕ С ДРУГИМИ ПАРАМЕТРАМИ

Параметр **src** совместим с такими параметрами:

- backup
- config
- defaults
- save (но у самого save в Ansible 2.2 проблемы с работой)

МОДУЛЬ IOS_CONFIG

ПАРАМЕТР NTC-ANSIBLE

NTC-ANSIBLE

ntc-ansible - это модуль для работы с сетевым оборудованием, который не только выполняет команды на оборудовании, но и обрабатывает вывод команд и преобразует с помощью TextFSM

Этот модуль не входит в число core модулей Ansible, поэтому его нужно установить.

NTC-ANSIBLE

Но прежде нужно указать Ansible, где искать сторонние модули. Указывается путь в файле ansible.cfg:

```
[defaults]

inventory = ./myhosts

remote_user = cisco
ask_pass = True

library = ./library
```

NTC-ANSIBLE

После этого, нужно клонировать репозиторий ntc-ansible, находясь в каталоге library:

```
[~/pyneng_course/chapter15/library]
$ git clone https://github.com/networktocode/ntc-ansible --recursive
Cloning into 'ntc-ansible'...
remote: Counting objects: 2063, done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 2063 (delta 1), reused 0 (delta 0), pack-reused 2058
Receiving objects: 100% (2063/2063), 332.15 KiB | 334.00 KiB/s, done.
Resolving deltas: 100% (1157/1157), done.
Checking connectivity... done.
Submodule 'ntc-templates' (https://github.com/networktocode/ntc-templates) registered for
Cloning into 'ntc-templates'...
remote: Counting objects: 902, done.
remote: Compressing objects: 100% (34/34), done.
remote: Total 902 (delta 16), reused 0 (delta 0), pack-reused 868
Receiving objects: 100% (902/902), 161.11 KiB | 0 bytes/s, done.
Resolving deltas: 100% (362/362), done.
Checking connectivity... done.
Submodule path 'ntc-templates': checked out '89c57342b47c9990f0708226fb3f268c6b8c1549'
```

NTC-ANSIBLE

А затем установить зависимости модуля:

```
pip install ntc-ansible
```

NTC-ANSIBLE

Так как в текущей версии Ansible уже есть модули, которые работают с сетевым оборудованием и позволяют выполнять команды, из всех возможностей ntc-ansible, наиболее полезной будет отправка команд show и получение структурированного вывода. За это отвечает модуль ntc_show_command.

NTC_SHOW_COMMAND

Модуль использует netmiko для подключения к оборудованию (netmiko должен быть установлен) и, после выполнения команды, преобразует вывод команды show с помощью TextFSM в структурированный вывод (список словарей).

Преобразование будет выполняться в том случае, если в файле index была найдена команда и для команды был найден шаблон.

NTC_SHOW_COMMAND

Параметры для подключения:

- **connection** - тут возможны два варианта: ssh (подключение netmiko) или offline (чтение из файла для тестовых целей)
- **platform** - платформа, которая существует в index файле (library/ntc-ansible/ntc-templates/templates/index)
- **command** - команда, которую нужно выполнить на устройстве
- **host** - IP-адрес или имя устройства
- **username** - имя пользователя
- **password** - пароль
- **template_dir** - путь к каталогу с шаблонами (library/ntc-ansible/ntc-templates/templates

NTC_SHOW_COMMAND

Пример playbook 1_ntc_ansible.yml:

```
- name: Run show commands on router
  hosts: 192.168.100.1
  gather_facts: false
  connection: local

  tasks:

    - name: Run sh ip int br
      ntc_show_command:
        connection: ssh
        platform: "cisco_ios"
        command: "sh ip int br"
        host: "{{ inventory_hostname }}"
        username: "cisco"
        password: "cisco"
        template_dir: "library/ntc-ansible/ntc-templates/templates"
      register: result

    - debug: var=result
```

NTC_SHOW_COMMAND

Результат выполнения playbook:

```
$ ansible-playbook 1_ntc-ansible.yml
```

```
SSH password:

PLAY [Run show commands on router] ********************************************

TASK [Run sh ip int br] *******************************************************
ok: [192.168.100.1]

TASK [debug] ******************************************************************
ok: [192.168.100.1] => {
    "result": {
        "changed": false,
        "response": [
            {
                "intf": "Ethernet0/0",
                "ipaddr": "192.168.100.1",
                "proto": "up",
                "status": "up"
            },
            {
                "intf": "Ethernet0/1",
                "ipaddr": "192.168.200.1",
                "proto": "up",
                "status": "up"
            },
            {
                "intf": "Ethernet0/2",
                "ipaddr": "unassigned",
                "proto": "down",
                "status": "administratively down"
            },
            {
                "intf": "Ethernet0/3",
                "ipaddr": "unassigned",
                "proto": "up",
                "status": "up"
            },
            {
                "intf": "Loopback0",
                "ipaddr": "10.1.1.1",
                "proto": "up",
                "status": "up"
            }
        ],
        "response_list": []
    }
}

PLAY RECAP ********************************************************************
192.168.100.1              : ok=2    changed=0    unreachable=0    failed=0
```

NTC_SHOW_COMMAND

В переменной response находится структурированный вывод в виде списка словарей. Ключи в словарях получены на основании переменных, которые описаны в шаблоне library/ntc-ansible/ntc-templates/templates/cisco_ios_show_ip_int_brief.template (единственное отличие - регистр):

```
Value INTF (\S+)
Value IPADDR (\S+)
Value STATUS (up|down|administratively down)
Value PROTO (up|down)

Start
  ^${INTF}\s+${IPADDR}\s+\w+\s+\w+\s+${STATUS}\s+${PROTO} -> Record
```

NTC_SHOW_COMMAND

Для того, чтобы получить вывод про первый интерфейс, можно поменять вывод модуля debug, таким образом:

```
- debug: var=result.response[0]
```

Пример playbook 2_ntc_ansible_save.yml с сохранением результатов команды:

```
- name: Run show commands on routers
  hosts: cisco-routers
  gather_facts: false
  connection: local

  tasks:

    - name: Run sh ip int br
      ntc_show_command:
        connection: ssh
        platform: "cisco_ios"
        command: "sh ip int br"
        host: "{{ inventory_hostname }}"
        username: "cisco"
        password: "cisco"
        template_dir: "library/ntc-ansible/ntc-templates/templates"
      register: result

    - name: Copy facts to files
      copy:
        content: "{{ result.response | to_nice_json }}"
        dest: "all_facts/{{inventory_hostname}}_sh_ip_int_br.json"
```

СОХРАНЕНИЕ РЕЗУЛЬТАТОВ ВЫПОЛНЕНИЯ КОМАНДЫ

Результат выполнения:

```
$ ansible-playbook 2_ntc-ansible_save.yml
```

СОХРАНЕНИЕ РЕЗУЛЬТАТОВ ВЫПОЛНЕНИЯ КОМАНДЫ

```
SSH password:

PLAY [Run show commands on routers] ********************************************

TASK [Run sh ip int br] ********************************************************
ok: [192.168.100.3]
ok: [192.168.100.1]
ok: [192.168.100.2]

TASK [Copy facts to files] *****************************************************
changed: [192.168.100.2]
changed: [192.168.100.1]
changed: [192.168.100.3]

PLAY RECAP *********************************************************************
192.168.100.1              : ok=2    changed=1    unreachable=0    failed=0
192.168.100.2              : ok=2    changed=1    unreachable=0    failed=0
192.168.100.3              : ok=2    changed=1    unreachable=0    failed=0
```

СОХРАНЕНИЕ РЕЗУЛЬТАТОВ ВЫПОЛНЕНИЯ КОМАНДЫ

В результате, в каталоге all_facts появляются соответствующие файлы для каждого маршрутизатора. Пример файла all_facts/192.168.100.1_sh_ip_int_br.json:


```
[
    {
        "intf": "Ethernet0/0",
        "ipaddr": "192.168.100.1",
        "proto": "up",
        "status": "up"
    },
    {
        "intf": "Ethernet0/1",
        "ipaddr": "192.168.200.1",
        "proto": "up",
        "status": "up"
    },
    {
        "intf": "Ethernet0/2",
        "ipaddr": "unassigned",
        "proto": "down",
        "status": "administratively down"
    },
...
```

ШАБЛОНЫ JINJA2

Для Cisco IOS в ntc-ansible есть такие шаблоны:

```
cisco_ios_dir.template
cisco_ios_show_access-list.template
cisco_ios_show_aliases.template
cisco_ios_show_archive.template
cisco_ios_show_capability_feature_routing.template
cisco_ios_show_cdp_neighbors_detail.template
cisco_ios_show_cdp_neighbors.template
cisco_ios_show_clock.template
...
```

ШАБЛОНЫ JINJA2

Список всех шаблонов можно посмотреть локально, если ntc-ansible установлен:

```
ls -ls library/ntc-ansible/ntc-templates/templates/
```

Или в репозитории проекта.

ШАБЛОНЫ JINJA2

Используя TextFSM можно самостоятельно создавать дополнительные шаблоны.

И, для того, чтобы ntc-ansible их использовал автоматически, добавить их в файл index (library/ntc-ansible/ntc-templates/templates/index)