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.

```
$ pip install ansible
```

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:

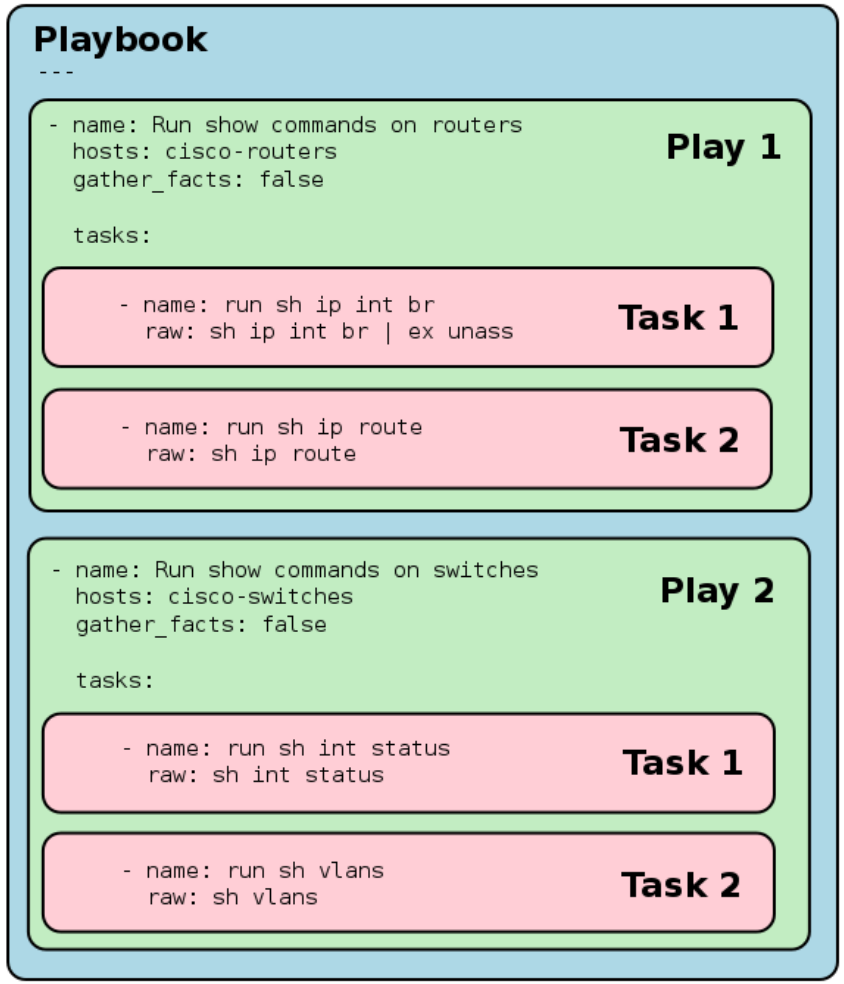
- play - это набор задач, которые нужно выполнить для группы хостов
- task - это конкретная задача. В задаче есть, как минимум:
  - описание (название задачи можно не писать, но очень рекомендуется)
  - модуль и команда (действие в модуле)

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
```

```
SSH password:

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

TASK [run sh ip int br] ********************************************************************************
changed: [192.168.100.2]
changed: [192.168.100.3]
fatal: [192.168.100.1]: FAILED! => {"changed": true, "failed": true, "msg": "non-zero return code", "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 vlan] *************************************************************************************
changed: [192.168.100.100]
        to retry, use: --limit @/home/vagrant/repos/pyneng-examples-exercises/examples/15_ansible/2_playbook_
basics/1_show_commands_with_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/chapter15/1_show_commands_
```

```
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:

```
ansible_connection: network_cli
ansible_network_os: ios
ansible_user: cisco
ansible_password: cisco
ansible_become: yes
ansible_become_method: enable
ansible_become_pass: cisco
```

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)

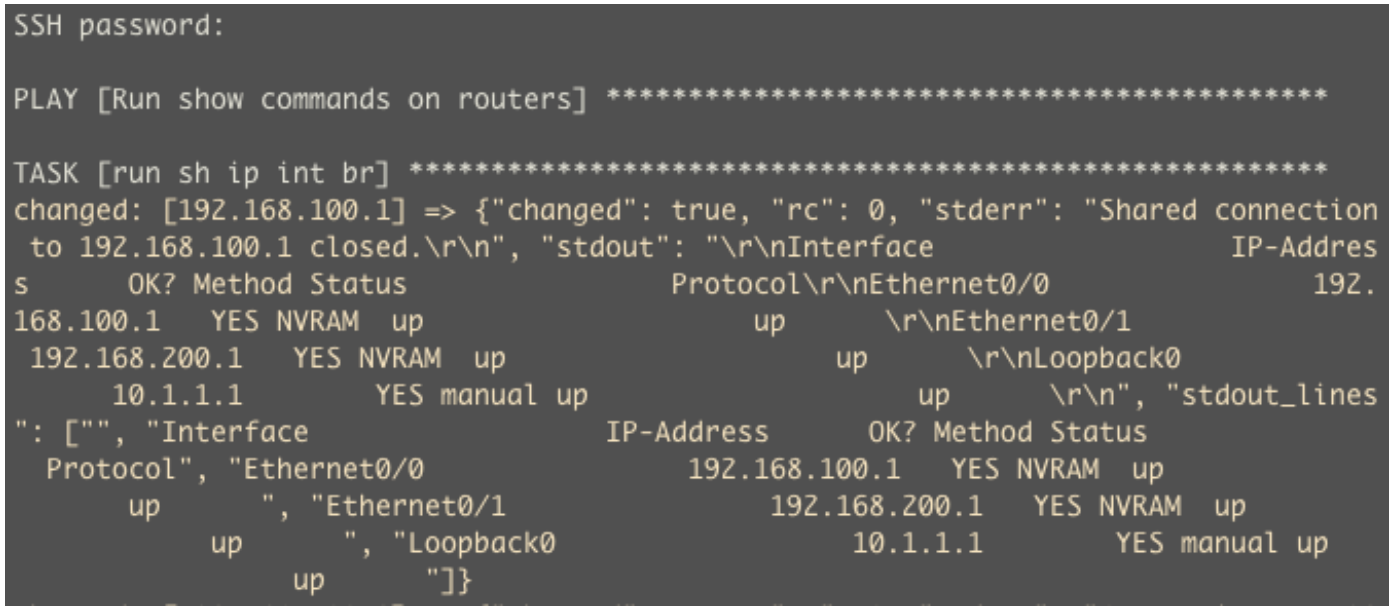
РАБОТА С РЕЗУЛЬТАТАМИ ВЫПОЛНЕНИЯ МОДУЛЯ

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
```



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
```